This is actually the first page of the thesis and will be discarded after the print out. This is done because the title page has to be an even page. The memoir style package used by this template makes different indentations for odd and even pages which is usually done for better readability.

Interactive Transformation Path Generation in Enterprise Architecture Planning

Philipp Immanuel Diefenthaler

Dissertation
for the degree of Doctor of Natural Sciences (Dr. rer. nat.)

University of Augsburg
Department of Computer Science
Software Methodologies for Distributed Systems

Supervisor:	**Prof. Dr. Bernhard Bauer**, Institute of Computer Science, University of Augsburg, Germany
Advisor:	**Prof. Dr. Wolfgang Reif**, Institute of Computer Science, University of Augsburg, Germany
Thesis Defense:	February 1st, 2016

dedicated to my friends, my family and Elina

Acknowledgements

First and foremost I thank my supervisor, Prof. Dr. Bernhard Bauer, for giving me the opportunity to conduct my research at the Software Methodologies for Distributed Systems (SMDS) as an external Ph.D. student. Furthermore, I thank him for his constant support and encouragement during the writing of this thesis, including his constructive backing in the tough times of the thesis. His comments and suggestions were and are always a valuable input for me.

I also thank Prof. Dr. Wolfgang Reif who accepted to be an advisor for my thesis.

Furthermore, I thank Dr. Florian Lautenbacher for his support that inspired and challenged me and my thesis not only in technical ways. Additionally, I thank him very much for his efforts in proofreading the thesis in an earlier version that helped to improve it a lot.

I am very grateful to the colleagues at the SMDS lab for many interesting discussions, a good mood and fun even though I was an external. Especially, I thank Melanie Langermeier for her great input and feedback in all discussions and having a great time at the ICEIS 2013 & 2014 and PoEM 2013 conferences.

I also thank my colleagues at Softplant that allowed me to conduct the research and gave me innovative support wherever possible. Additionally, I am very grateful for the opportunity to present and discuss my research at conferences all over Europe.

Special thanks goes to my family that gave me all the things that allowed me to successfully finish my thesis and built the basis for what is today me. This includes all my friends that have become part of my family over the past years. You are awesome. Furthermore, I thank you all for your patience in all the hours I spent with the thesis and not with you.

Last but not least I would like to thank Elina that was always supportive with her happy and powerful attitude. Furthermore, I thank her for her patience in all the hours I spent with the thesis and not with her.

Abstract

Nowadays, business and information technology (IT) of an enterprise need to co-evolve with an increasing pace in order to stay competitive. New markets and competitor shifts force enterprises to react quickly to changes and realize adequate support through the IT. However, the application landscapes of enterprises are heterogeneous, technically outdated and hardly cope with the desired speed of change of the business.

Enterprise architecture management and its respective planning discipline provides a holistic approach for improving the accompanied change of business and IT in favor for both domains. Through the creation of a model of the current enterprise architecture, current architecture for short, a common understanding of the interrelationships between stakeholders of the business and IT domain and their respective domain artifacts is established. Different target enterprise architectures, target architecture for short, allow to develop different possible future states of the enterprise and its business and IT. Afterwards, it is possible to decide for one of the different transformation paths to realize the target architecture in a purposeful way, as those changes are prone to resource bottlenecks.

Many approaches provide solutions for (meta-)modeling and methodological aspects. However, decision support of enterprise architects by sophisticated means from decision theory, automatic model exploration and ranking mechanisms is still scarce.

Therefore, the thesis at hand develops an interactive approach for the generation of transformation paths to assist enterprise architects in their activities through a combination of a multi-criteria decision making technique, a graph formalism based automated planner and a formalization of the possible changes. We call the latter transformation actions. The approach is based on three pillars that we will briefly present in the following.

Firstly, we introduce the necessary underlying formalisms and model fragments to allow an automated planner to discover possible future target architectures. Furthermore, we present requirements that are to be considered by our approach and a mechanism to abstract from modeled situations to a common pattern. We call the latter transformation patterns.

Secondly, we present our approach for decision support in target architecture selection in the first phase of the planning effort. Additionally, we provide the transformation actions and means to rank them that allow an enterprise architect to interactively select them.

Thirdly, we introduce the approach for the actual decision upon the sequence in which the changes are to be realized in the second phase. We allow for a reuse of results from our former phase. Besides, we decouple the phases by just postulating necessary information to start the second phase without limiting the approach for the first phase to ours. Furthermore, we present the transformation actions and how the foundations for the creation of a roadmap are set to allow for a decision upon changes and their enactment.

Kurzfassung

Um in der heutigen Zeit wettbewerbsfähig zu bleiben, muss das Geschäftsmodell und die Informationstechnologie im Unternehmen gemeinsam weiterentwickelt werden. Neue Märkte und Veränderungen der Wettbewerbssituation verlangen von Unternehmen ein schnelle Reaktion auf diese Veränderungen und eine angemessene Unterstützung durch die IT. Heutige Anwendungslandschaften in Unternehmen sind zumeist heterogen, technisch veraltet und werden den Anforderungen an die Umsetzungszeit der Veränderungen nur selten gerecht.

Unternehmensarchitekturmanagement und seine zugehörige Planungsdisziplin bietet einen ganzheitlichen Ansatz zur gleichzeitigen Verbesserung des Geschäftsmodells und der IT im Rahmen von Veränderungen. Mit Hilfe eines Modells der Ist-Unternehmensarchitektur wird ein gemeinsames Verständnis zu den Zusammenhängen, zwischen den Interessensgruppen der Geschäfts- und IT-Welt, sowie die für diese Domänen relevanten Artefakte, gewonnen. Unterschiedliche Soll-Unternehmensarchitekturen erlauben die Entwicklung unterschiedlicher, möglicher Zustände des Unternehmens, sowie seiner Geschäfts- und IT-Welt. Danach ist es möglich sich zu entscheiden welchen Transformationspfad man zur Erreichung der Soll-Architektur sinnvollerweise wählt, da die Veränderungen anfällig für Ressourcenengpässe sind.

Viele Ansätze bieten Lösungen für die (Meta-)Modellierung und methodische Themen. Die Entscheidungsunterstützung für Unternehmensarchitekten mit Hilfe von weitreichenderen Lösungen, die sich aus der Entscheidungstheorie, der automatischen Modellfindung und Mechanismen zur Bewertung zusammensetzen ist jedoch immer noch selten.

Deshalb widmet sich die vorliegende Arbeit einem interaktiven Ansatz für die Erstellung von Transformationspfaden die Unternehmensarchitekten bei ihren Aufgaben mittels einer multikriteriellen Entscheidungstechnik, einem graphbasierten automatischen Planer und der Formalisierung möglicher Veränderungen unterstützt. Letztere nennen wir Transformationsaktionen. Der Ansatz basiert auf drei Säulen, die wir im folgenden kurz ausführen.

Zuerst stellen wir die notwendigen unterliegenden Formalismen und Modellfragmente vor, die es dem automatischen Planer erlauben, mögliche zukünftige Soll-Architekturen zu entdecken. Des Weiteren stellen wir Anforderungen vor, die von unserem Ansatz erfüllt werden müssen und einem Mechanismus, der es uns erlaubt von unterschiedlich

modellierten Situation ein gemeinsames Muster abzuleiten. Letztere nennen wir Transformationsmuster.

Als zweites stellen wir den Ansatz zur Entscheidungsunterstützung bei der Auswahl einer Soll-Architektur in der ersten Planungsphase vor. Zusätzlich stellen wir die Transformationsaktionen und deren Bewertungsmechanismus vor, der es Unternehmensarchitekten erlaubt diese interaktiv auszuwählen.

Drittens stellen wir unseren Ansatz für die zweite Planungsphase vor, mit dessen Hilfe die tatsächliche Auswahl der Abfolge in der die Veränderungen ausgeführt werden sollen, bestimmt wird. Wir erlauben die Wiederverwendung der Ergebnisse aus der vorhergehenden Planungsphase. Trotzdem sind beide Phasen entkoppelt, indem notwendige Informationen für die zweite Phase gekennzeichnet sind, ohne sich dabei auf unseren Ansatz aus der ersten Phase zu beschränken. Außerdem präsentieren wir die notwendigen Transformationsaktionen und zeigen wie die Grundlagen zur Erstellung einer Roadmap geschaffen werden, die dann die Umsetzung der Veränderungen festlegt.

Contents

1 Introduction

The efficient and effective management of an enterprises' information technology (IT) as an enabler for competitive advantage, but also as a cost center has become a major challenge. As the interconnection of strategic goals, applications, processes, organization units, and infrastructure components is complex and changes within one domain impact elements of others a holistic approach is needed. Enterprise architecture management (EAM) provides such a holistic view on the enterprise by connecting the different concepts and providing support for the management of them. The management process of EAM follows a plan, do, check, act cycle (Buckl and Schweda, 2011). The planning phase of the cycle is concerned with the assessment of the current situation and future anticipated change demands. Enterprise architecture (EA) models, which represent different states of the architecture for different points in time, support the planning process. A current architecture serves as the planning basis. A target architecture serves as a desired future state. They are modeled to derive necessary changes. Several different target architectures can be modeled to compare alternative developments.

The remaining phases of the cycle are concerned with employing the intended changes in the enterprise (do), measuring the progress of the employed changes against the intended (check), and adapting the previously planned targets due to the new insights gained in the employment of changes and due to the changed situation of the enterprise (act).

However, these models are not sufficient to plan transformation paths, that optimally change the IT from the current towards a future state. A transformation path consists of transformation actions, which change the current EA into the target EA. Deciding which alternative transformation actions are the best, in terms of cost, time, and risk of changes and in which sequence they should be pursued is a complex task. Furthermore, generating possible transformation paths poses additional challenges, because it is necessary to consider dependencies between elements of different states and the interdependencies of changes. Additionally, depending on the current situation of the enterprise different preferences on the transformation actions have to be considered.

A transformation model, which contains information on the successor relationships between elements of different states, was already introduced to support the transformation path generation (Aier and Gleichauf, 2010a). However, the derivation of sequences of changes and the ranking of them remains a manual task. Furthermore, using a planning domain model for applying automated planning techniques as a means to explore

possible target architectures and the actual paths was not investigated yet. Such a planning domain model is necessary to interactively support the enterprise architect in the decision making and her modeling activities.

By interactive support we mean the possibility of an enterprise architect to easily express her preferences and test impacts on the ranking of the alternatives by changing the preferences. Furthermore, she gains support by automatic model creation of alternative target architectures and obtains feedback regarding the efforts of different pursuable transformation paths.

1.1 Problem Statements and Challenges

In the following we introduce the different topics and their identified problem statements and challenges. The problem statements and challenges introduced below are important for the design of a solution to allow for an interactive generation of the transformation path.

Transformation Patterns and Planning Domain Model

Enterprise architecture models for transformation planning and their characteristics are the foundation to allow for a formal description of the planning domain. To be able to support the path decisions we need a mechanism for abstraction that allows for deriving commonalities of modeled situations. We call this mechanism of abstraction transformation patterns. Furthermore, they allow for an interactive transformation path generation.

Problem Statement. Up to now there is no sound formal description of the planning domain for enterprise architecture models available. However, this is necessary for planning transformation paths semi-automatically, because they impose temporal dependencies and restrictions on the changes from the model of the current enterprise architecture to a target architecture. Furthermore, information underlying the different views which are generated for planning purposes is implicit. This is, because changes are depicted for two different points in time and leave space for interpretation for the time between both states. From a practical point of view the current models provide only static aspects in terms of states, which should be complemented with means to improve the decision making and the overall success of transformations.

Challenges. To develop a planning domain model which is compatible with the major EA metamodels is a challenge because EA models differ in elements, relationships

and attributes. However, it is necessary to establish this compatibility to ensure the acceptance of the planing domain model after its design. Extensions to existing EA metamodels to conform to the requirements of the planing domain model are documented as well. Furthermore, EA models for planning differ in level of detail and the underlying assumptions which elements, relationships and attributes are considered as part of the current and target architectures vary. They differ in detail because enterprise architects use different methods to plan the target architecture, which may produce a different level of detail and use different concepts to be planned. The transformation patterns allow to focus on certain types of situations that we will support in an interactive way. By introducing the planning domain model we consider a limited set of transformation patterns.

Decision Support for Target Architecture Selection

Enterprise architecture management uses visualizations for analysis, planning and communication purposes. Regarding planning activities business support maps are a common means to express changes in the business IT-alignment. Business support maps belong to the group of cartesian visualizations. They allow to model a current business support and a desired business support of the business through the IT for different points in time. Using visualizations, like business support maps, already has an impact on the number of possible transformation paths that may be pursued. Therefore, it is important to support at first the selection of a target architecture to be able to generate the transformation paths.

Problem Statement. Business support maps only capture states and do not provide information how a transformation takes place. Furthermore, a common means in business support maps is the use of logical planning entities, that are implementation independent. Specifications of logical entities do not define a target architecture with concrete applications and application services. However, a transformation is always applied to elements which lifecycle phases' can change due to transformation efforts. Additionally, the enterprise architect is currently not supported by automatic means to derive rankings for changes, that would aid in transformation decisions. Such information should support the decision regarding the resource allocation that should be used to perform a certain transformation action.

Challenges. The use of logical elements in the business support maps poses a challenge to the modeling support of target architectures because they span a decision space which contains alternatives with differing benefits and drawbacks. As a result the formalization and automatic derivation of the factors influencing the decision making for a target enterprise architecture poses a challenge for interactive transformation path generation. The factors have to consider cost, time, and risk for different modes of resource allocation.

Transformation Path Generation

The generation of the sequence of transformation actions is considered in the process that creates the transformation path. Given a current architecture, target architecture and the transformation model there are different transformation paths, which could be pursued.

Problem Statement. Given a current and target architecture there are several different transformation paths which are implicit, even though there is a transformation model. This is also true for the changes of dependencies between elements of the enterprise architecture. Assume the following situation: there exists an explicit dependency between predecessor elements which no longer holds in the target enterprise architecture for their successor elements. There is no statement if there is a temporal restriction on the change of the dependency, as it is neither explicit in the current nor target enterprise architecture. Furthermore, the retirement of old elements may be hindered by the creation of new elements, which are supposed to replace the old elements.

Challenges. The challenge in the generation of the transformation path is that the changes lead from one initial state to one goal state in a confluent way. Furthermore, the consideration of reference scenarios, which introduce temporary objects and dependencies, for risk mitigation in the transformation path pose a challenge. Reference scenarios are a description of a best practice, which should be used in the transformation, if a certain kind of transformation is planned, e.g. the split of an application into a purely core processes supporting application and into a purely supporting processes facilitating application. Additionally, ranking the sequences needs to consider the potential future benefits of the effect a change has, as a costly change might enable the benefits of several beneficial changes.

1.2 Objectives, Approach, and Contributions

In the following section we define the research objectives for the presented topics. The objectives are a response to the posed problem statements and challenges. Furthermore, we describe the approach used to design our solution and its contributions. Figure 1.1 shows an overview of the topics and important concepts in context.

At the bottom of Figure 1.1 the foundational concepts important for decision support when selecting a target architecture and as well as transformation path generation are depicted. The planning domain model is the underlying metamodel that allows to consider certain transformation patterns that abstract from modeled situations in a current, a target architecture and their corresponding transformation model.

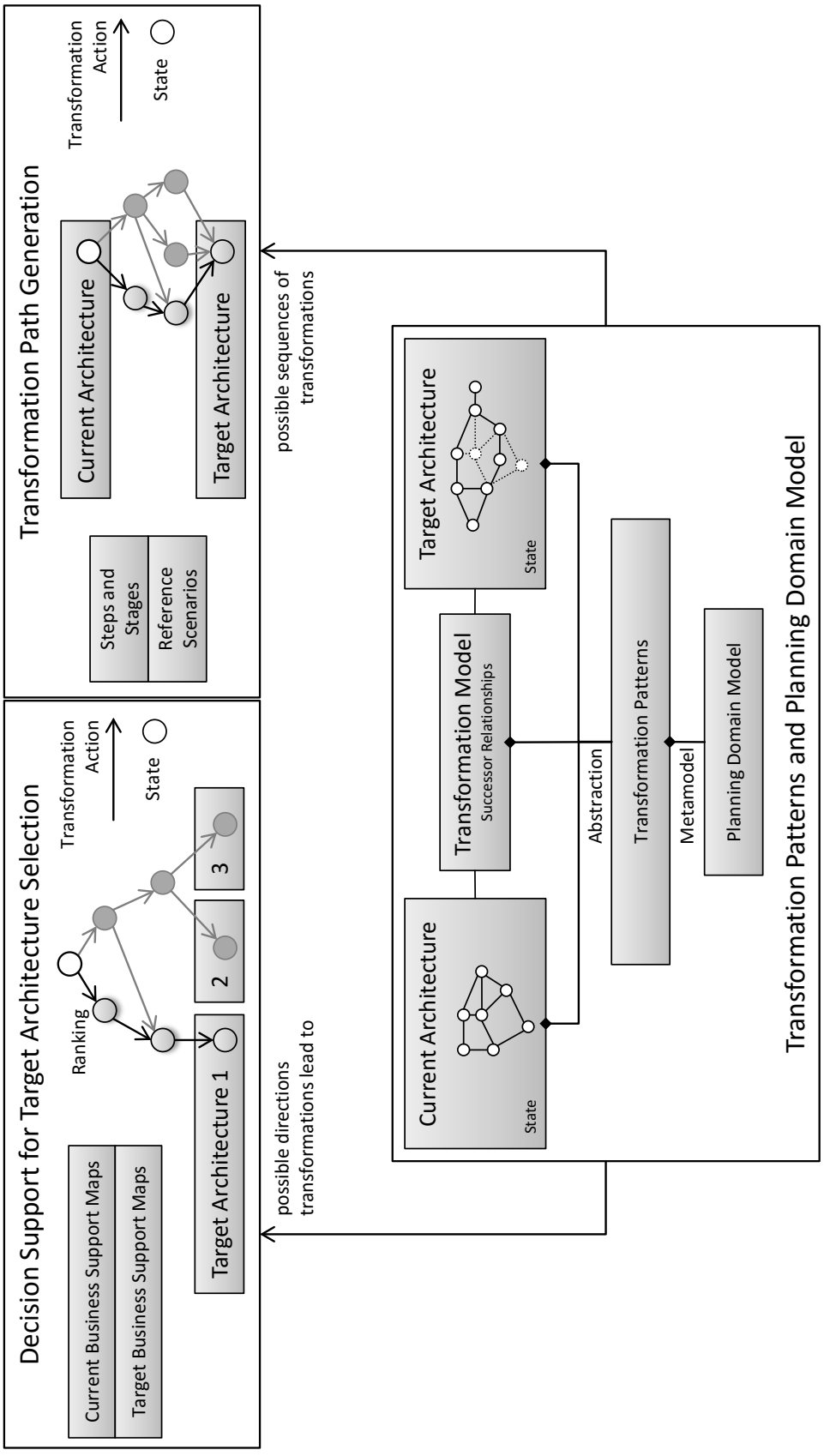

Figure 1.1: Topics and Important Concepts in Context

The top left part of Figure 1.1 shows the foundational concepts for the decision support involved in selecting a target architecture. Current and target business support maps are used as means to support the decisions regarding possible reachable target architectures. A ranking for transformation actions is provided to support the decision making. Each application of a transformation actions leads to another state of the architecture.

At the top right of Figure 1.1 the foundational concepts for the generation of transformation paths are depicted. The different possible sequences from a current to a target architecture are considered here. Reference scenarios allow to consider preferred detours, neither specified in a current nor target architecture, in the transformation paths. Furthermore, it is possible to create steps and stages that serve as the foundation for roadmaps later on to enact the transformation.

1.2.1 Transformation Patterns and Planning Domain Model

As the overall goal of the thesis is to allow for an interactive planning of transformation paths it is necessary to provide adequate formalisms that enable an automated planner to support the enterprise architect. However, in order to be able to choose such formalisms it is necessary to provide means for abstractions from situations to commonalities present in the models. Therefore, we formulate our first research objective.

Objective 1. Enable a formalized abstraction of the commonalities in EA models involved in transformation path planning via transformation patterns and derive a planning domain model.

Approach. The design process for planning domain models from Vaquero et al. (2011) is used for the development of the planning domain model which allows to encode the transformation patterns within a formal metamodel. After establishing a common understanding on EA models used for transformation planning we present a conceptualization of transformation patterns by using a metamodel independent view on architecture states for planning transformation paths. The transformation patterns are narrowed down by scoping the concerns to be considered in the transformation paths. As a result we present the requirements for the planning domain model that are the first phase of the design process for planning domain models. Furthermore, we present a formalization of the planning domain model using a type graph.

Contributions. Based on the concept of transformation patterns and the concerns to be considered in the planning process of transformation paths we derive requirements for a planning domain model. The designed artifacts are the planning domain model, a mapping of the planning domain model to existing EA metamodels, a set theoretical description of the subsets of EA models for different points in time, and the implications for the artifacts of the following contributions.

1.2.2 Decision Support for Target Architecture Selection

Regarding the number of possible target architectures that could be realized we need to support at first the decision process for one of them. After selecting a target architecture it is possible to sequence the transformation actions. Therefore, we formulate our second research objective.

Objective 2. Enable interactive decision support for selecting a target architecture considering efforts of changes and allow for an automatic creation of a transformation model.

Approach. We combine a multi-criteria decision making technique to provide a ranking with an automatic exploration of target architectures via graph transformations. For calculating the resulting efforts of the transformation actions we reuse simplified project management dimensions used for project planning. As a planning mechanism for selecting a target architecture we use business support maps.

Contributions. Our contribution contains an approach which interactively supports an enterprise architect in the creation of a target enterprise architecture using business support maps. We realize the approach by automatically generating possible target enterprise architectures, calculating the estimated efforts for their realization. An enterprise architect then chooses to remove or add changes, according to a list of ranked alternatives. The architect can overrule suggestions which are automatically generated. The preferences regarding the criteria of the efforts are made transparent in the decision making process.

1.2.3 Transformation Path Generation

After selecting a target architecture it is necessary to support the enterprise architect in her decisions regarding the order in which the transformation actions are to be conducted. In contrast to decision support for target architecture selection, that allows to determine the target architecture in detail and a corresponding transformation model, the process of generating the transformation path is concerned with determining the sequence of transformation actions from the current to the target architecture. Therefore, we formulate our third research objective.

Objective 3. Enable interactive transformation path generation given a current architecture, target architecture and the corresponding transformation model.

Approach. For the exploration of possible sequences of transformation paths we use graph transformations. Steps and stages are derived from a graph transformation system,

which becomes explorable through the graph transformations. This mechanism allows the enterprise architect to determine transformation actions to be executed in parallel. Furthermore, we allow for the automatic consideration of reference scenarios introducing temporary elements and relationships in the generation of the transformation paths.

Contribution. We provide a formalization of reference scenarios, a method to derive steps and stages within the transformation path. Furthermore, we allow for the interactive creation of transformation paths using transformation actions. The optimal transformation path is figured out interactively with the enterprise architect, as some of her experiences are not formalized due to cost benefit considerations.

1.2.4 Demonstration and Evaluation

The demonstration and evaluation of a new approach is important to show its practicability and benefits, but also its limitations. Regarding the demonstration it is important to show how the approach is realized. In contrast the evaluation focuses on the determination of how satisfactory the objectives were reached and points out limitations of our approach. Therefore, we set a forth objective for the thesis at hand.

Objective 4. Create an instantiation of the approach and evaluate it.

Approach. We present a methodology to develop, test, maintain, and extend a transformation action repository that contains all transformation actions. Our approach is evaluated with three use cases and a scenario based evaluation. The scenario-based evaluation is used in software engineering to evaluate software architectures (Zhu, 2005). We adapt the evaluation techniques for evaluating our approach.

Contribution. We provide an instantiation of our approach to show its feasibility regarding the posed problem statements and challenges. Furthermore, we point out limitations and discuss them.

1.3 Research Methodology Applied in the Thesis

Overall the thesis follows a design science approach, which was introduced by Hevner et al. (2004). Design science research solves relevant problems, by reusing existing knowledge that is suitable for the problems posed and creating artifacts. The design creates those purposeful artifacts that are either models, methods, constructs, and instantiations or a combination thereof. To be able to communicate and define the problems and their solutions the constructs are used. The models reuse them to create a connection

between the problem and the solution. In contrast methods allow to define processes to create the connection and limit the number of possible solutions. To demonstrate the feasibility of the designed artifacts instantiations of them are created.

For conducting the research we followed the design science research process as proposed from Peffers et al. (2006), that consists of the following phases:

1. Problem identification & Motivation
2. Objectives of a solution
3. Design & Development
4. Demonstration
5. Evaluation
6. Communication

The first phase is concerned with scoping the problem and motivating its relevance. Within the second phase the objectives of a solution for the problems at hand are defined. The actual design of the artifacts and their development is considered in the third phase. A demonstration to show that the solution solves the problems in an suitable context is done in phase four. The fifth phase evaluates the designed artifacts regarding their efficiency and effectiveness. In the last phase the communication of artifacts and results takes place to gain further feedback and to make the newly created knowledge accessible for other researchers. The phases are sequenced as introduced above, but it is possible to reiterate the phases if necessary.

1.4 Publications

Parts of the thesis were published previously in different publications. Some of the concepts and approaches presented in these publications have evolved due the feedback of the reviewers and the audience at the conferences. The publications are listed chronologically below:

1. Chen, W., Hess, C., Langermeier, M., Stülpnagel, J. v., and Diefenthaler, P. (2013). Semantic Enterprise Architecture Management. In *15th International Conference on Enterprise Information Systems (ICEIS 2013)*, pages 318–325.

   This publication describes semantic web technologies to facilitate a semi-automatic

documentation of EA models by linking different model repositories from different domains, like business process modeling and application systems modeling. Furthermore, the publication describes how a reasoner allows to check the consistency of the EA model and how implicit knowledge within the EA model is reasoned. This paper shows that is possible to use semantic web technologies for creating EA models that ensure a certain degree of formality and serves as a basis for the models within Chapter 3 of the thesis at hand. The author of this thesis participated in the design of the ontologies and queries.

2. Diefenthaler, P. and Bauer, B. (2013). Gap Analysis in Enterprise Architecture using Semantic Web Technologies. In *15th International Conference on Enterprise Information Systems (ICEIS 2013)*, pages 211–220.

   This paper presents an approach how the gap analysis is performed on EA models based on semantic web technologies. An advantage of the approach is that it is EA metamodel independent and that it can be reused at various stages within the EA planning process. Furthermore, it shows how a simple query language is used to automatically add the successor relationships between elements. This paper served as the basis for the concepts used in Chapter 3 and already showed the applicability for the Best–Practice EA - Use Case that is presented in detail in Section 7.3. The author of this thesis designed the approach and realized it with the valuable feedback of the co-author. The paper was awarded with the *Best Paper Award* in the *Enterprise Architecture Track* at the *15th International Conference on Enterprise Information Systems*.

3. Lautenbacher, F., Diefenthaler, P., Langermeier, M., Mykhashchuk, M., and Bauer, B. (2013). Planning Support for Enterprise Changes. In Grabis, J., Kirikova, M., Zdravkovic, J., and Stirna, J., editors, *The Practice of Enterprise Modeling*, volume 165 of *Lecture Notes in Business Information Processing*, pages 54–68. Springer, Berlin, Heidelberg.

   In this publication the usage of graph transformations for planning the transformation path from a current to a target architecture is presented. Besides, showing that it is possible to semi-automatically create transformation paths we gained insights to possible improvements. The insights were a missing ranking of different transformation actions, a lacking interactivity in the selection process of transformation actions and the hurdle of a target architecture with the same level of detail as the current architecture as a starting point for planning. Nevertheless, it builds the basis for the approach presented in Chapter 5 and it also introduced the feasibility of it in an early stage for the Development Master Data Management - Use Case, that is presented in detail in Section 7.4. The author of this thesis designed and realized the parts of the approach concerned with the planning activities and their formalization. Furthermore, he conducted the evaluation with the enterprise architect.

4. Diefenthaler, P. and Bauer, B. (2014a). From Gaps to Transformation Paths in Enterprise Architecture Planning. In Hammoudi, S., Cordeiro, J., Maciaszek, L. A., and Filipe, J., editors, *Enterprise Information Systems*, volume 190 of *Lecture Notes in Business Information Processing*, pages 474–489. Springer International Publishing, Cham.

   This article is an extended version of the paper that describes the gap analysis with semantic web technologies. Besides, claiming the need for a more advanced formalism to support the transformation path generation the article shows how it is possible to create transformation paths with a set of gaps. Event though the approach presented in the article does not mention the decision support for target architecture selection it build the origin for supporting these decisions in the first place. Therefore, it is the foundation for Chapter 4 and also influenced Chapter 5. The author of this thesis designed and realized the approach.

5. Diefenthaler, P. and Bauer, B. (2014b). Using Gap Analysis to Support Feedback Loops for Enterprise Architecture Management. In Kundisch, D., Suhl, L., and Beckmann, L., editors, *MKWI 2014 - Multikonferenz Wirtschaftsinformatik*, Paderborn.

   This paper presents a set theoretic conceptualization of three EA models for different points in time and from different points in time. Using these concepts a feedback loop is created to filter relevant information for an enterprise architect, in the retrospective of the EA management cycle, out of the models. It is called feedback loop viewpoint as the goal of the filter mechanism is to help the enterprise architect in gaining an overview of plan deviations, that she reuses in the next planning phase for improvements. This contribution mainly influenced the concepts presented in Section 3.3. The author of this thesis designed and realized the approach.

6. Ortmann, J., Diefenthaler, P., Lautenbacher, F., Hess, C., and Chen, W. (2014). Unternehmensarchitekturen mit semantischen Technologien. *HMD Praxis der Wirtschaftsinformatik*, 51(5):616–626.

   This article introduces the approach of establishing an EA repository using semantic web technologies. Furthermore, first results for supporting the enterprise architect in choosing a target architecture is presented. The approach presented in the article influenced the origin of transformation patterns. Those parts build the frame for the detailed description in Chapter 4. The author of this thesis designed and realized the parts of the approach concerned with the transformation planning.

7. Diefenthaler, P., Langermeier, M., and Bauer, B. (2015). Interactive Transformation Path Generation Using Business Support Maps. In Barjis, J. and Pergl,

R., editors, *Enterprise and Organizational Modeling and Simulation, 11th International Workshop, EOMAS 2015, Held at CAiSE 2015*, Lecture Notes in Business Information Processing, Heidelberg. Springer.

In this publication the approach for ranking transformation actions using the Weighted Product Model in a bottom-up planning scenario is presented. Given business support maps we describe how transformation patterns for the business support maps are derived and appropriate transformation actions are modeled. Furthermore, we describe how the approach is applied to the Living EA - Use Case. The approach described in the paper is elaborated in more detail in Section 4.2 and the Living EA - Use Case is evaluated in detail in Section 7.2. The author of this thesis designed and realized the approach with the valuable feedback of the co-authors.

1.5 Outline of the Thesis

In this chapter we have introduced the problem statements and interrelated challenges. We then derived the topics and objectives of the thesis. Figure 1.2 shows the outline of the thesis.

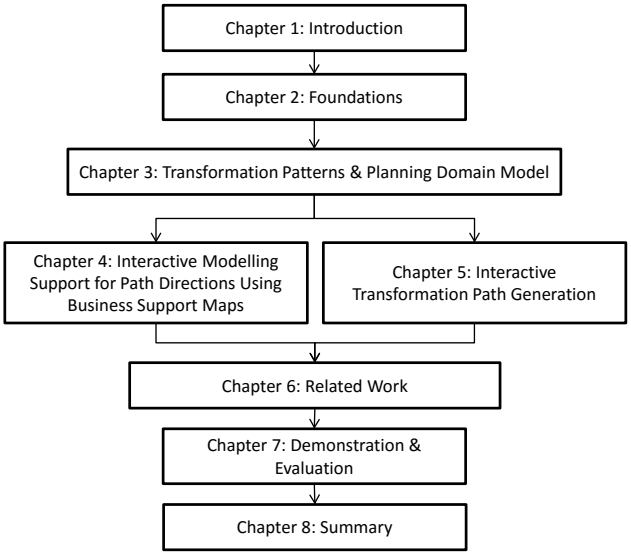

Figure 1.2: Outline of the Thesis

The thesis continues in Chapter 2 with the necessary foundations to understand the domain of enterprise architecture management and transformation paths. Furthermore, foundations necessary for the formal parts of the thesis and their domains are introduced.

In Chapter 3 we introduce transformation patterns and the planning domain model. The latter is the meta-model used for transformation path planning and is formalized as an attributed type graph. Whereas, transformation patterns allow for an abstraction of modeled situations that allow us to define necessary model fragments for the planning domain model. Besides, the chapter presents our understanding of the concepts relevant for transformation paths in EA planning and their formalization. This builds the basis for the approaches described in the subsequent chapters to interactively support planning transformation paths.

The interactive decision support for target architecture selection is introduced in chapter 4. We present an approach for bottom-up, as well as top-down planning. These approaches differ in the transformation actions to be used, the mechanism to rank these and the start states. However, they share some commonalities that we point out. Furthermore, we show, even though the approaches are different, how a target architecture is selected and how this task is supported by means of multi-criteria decision making technique.

Chapter 5 presents the approach for interactively generating transformation paths. After presenting the required models and input from an enterprise architect, we formalize the concepts for transformation path generation. This allows us to interactively explore possible paths from the current to the target architecture. Furthermore, we provide means to assign transformation actions to steps and stages, that are a crucial input to proceed with scheduling the changes.

We continue in Chapter 6 with related work for enterprise architecture management. Different approaches for modeling planning issues in enterprise architectures exist and are therefore presented and discussed. Furthermore, we present and discuss existing tool support for EAM. Additionally, we discuss existing optimization approaches in EA planning. The chapter ends with already existing pattern mechanisms in EAM and the distinction to the transformation patterns used in our approach.

We demonstrate and evaluate our designed artifacts in Chapter 7. We present the transformation action repository that allows us to develop, test, maintain, and extend the transformation actions provided for transformation path planning. Furthermore, we present three different use cases and scenarios for evaluation. The Living EA - Use Case allows for an evaluation of the bottom-up approach, whereas the Best–Practice EA - Use Case evaluates the top-down approach. The third use case is the DMDM - Use Case and allows to evaluate the actual transformation path generation for a development master data management in an industrial setting.

The thesis ends with a summary in Chapter 8. We discuss our conclusions on the presented approach and suggest future work that was identified due to the insights gained through the conducted research.

2 Foundations

In this chapter we introduce the background information that helps to better understand the main parts of the thesis. At first we introduce enterprise architecture management with a focus on transformation path generation and important existing frameworks. Afterwards, we present scientific techniques which allow to realize the interactive support for transformation path generation. These techniques are reused from the fields of semantic web based knowledge management, automated planning, and multi-criteria decision making. Furthermore, the foundations of graph grammars, narrowed to the topics important for this thesis, are introduced, as they are the foundational formalisms used in transformation path planning later on.

2.1 Enterprise Architecture Management

Enterprise Architecture Management (EAM) follows a typical management cycle that consists of the phases plan, do, check and act (Buckl and Schweda, 2011; Bucher et al., 2006; Hanschke, 2013; Niemann, 2006). The plan phase is concerned with developing change proposals that are implemented in the do phase. Therefore, EAM creates and updates documents that serve as the basis for the decisions upon changes by an architecture board. An architecture board is a group of people within the enterprise that comprises stakeholders from different domains which are impacted by architectural decisions but also may provide useful information for architectural changes (The Open Group, 2011, Chapter 47). The involvement of stakeholdes from different domains improves the commitment of departments to support the changes, but also makes the decision making process more time consuming as every stakeholder has different goals and building a consensus can be difficult. The do phase is concerned with the implementation of the changes which is supported in many enterprises by change management. Implementation in this context is not concerned with coding new applications, but with changing business processes, organizational structures and if necessary prepare employees to use new or change applications. Within the check phase differences between intended and actually achieved results are controlled. Based upon the results from the check phase the act phase provides input to the plan phase by supplying information for the next plan phase. Models of an enterprise, support the plan phase as part of EAM and are a special form of documentation (Aier and Gleichauf, 2010a; Buckl et al., 2009a).

The advantage of the models is that their visualization is supported by tools and allows to discuss issues with stakeholders based on a shared visual representation.

EA models are used to describe an EA for different points in time (Buckl and Schweda, 2011) and are often partitioned in architectural layers. Common layers cover aspects from business, application, information, technology, and cross-cutting issues. Figure 2.1 shows the architecture layers as suggested by Dern (2009), arranged as a pyramid. At the top the strategy layer considers issues like goals and objectives of the enterprise. They influence the evaluation and design of the aspects covered in the business layer. In this layer common elements are business processes, organization units, roles of employees, and products, including services, produced by the enterprise. Below the business architecture layer an information architecture layer is positioned, according to Dern (2009). It unifies the different views on business and IT, but at the same time decouples their elements for independent terminologies and development. Such a decoupling is called virtual decoupling, as elements are introduced that are just created and used to abstract from objects used in the adjacent layers, but with relevance for business and IT architecture (Aier and Winter, 2009). The IT architecture layer considers the applications (software) and their interconnections. Sometimes this layer is also called application landscape. Necessary technology to deploy the applications and allow their communication is considered in the IT infrastructure layer. The IT-Project portfolio layer considers cross-cutting aspects of projects. Sometimes such cross-cutting layers are also called vertical layers in contrast to horizontal layers. They may influence elements and relationships between them in different (horizontal) layers. A common considered further vertical layer, not depicted in Figure 2.1, is a security architecture layer as considered for example by Murer et al. (2011, Figure 2.10).

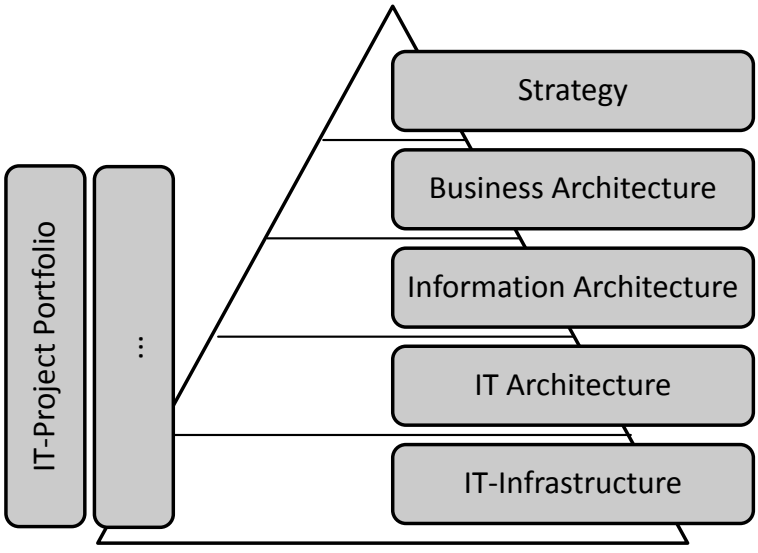

Figure 2.1: Possible Layers of an Enterprise Architecture; Adapted from Dern (2009, Figure 1-1)

Depending on the EAM approach and information needs of enterprise architects different element types, relationships, attributes, and layers are present in an EA metamodel, that formalizes the modeling possibilities. In general it is necessary to identify, clarify, and connect the concepts of the business and IT to create a common understanding between both disciplines. Furthermore, a clarification is necessary to check whether the meaning of an element type is the same for business and IT even if the name is the same (Glissmann and Sanz, 2011). If no clarification takes place EA models are prone to misinterpretation and lead to erroneous decisions. A further distinction can be made regarding the levels of detail in the EA model that are dependent on the time horizon used for planning (Pulkkinen, 2006; Hanschke, 2013; Goethals et al., 2006).

The model of the current architecture of the enterprise is a documented enterprise architecture at the present point in time and serves as a starting point for defining a model of a target architecture. In contrast, the model of the target architecture represents a desired architecture in the future which is used to guide the development of an EA from the current towards a target architecture. The development of a target architecture depends on the enterprises' goals. It is influenced by business requirements, strategic goals and IT objectives like master data consolidation, improving the flexibility of IT and drive the coverage of standard platforms (Hanschke, 2013).

The level of detail in EA models is dependent on the time horizon of the planning effort and is divided into strategic, tactical, and operational levels (Hanschke, 2013; Goethals et al., 2006; Greefhorst and Proper, 2011). Figure 2.2 shows the conceptual dependencies between the management levels, time horizons, and level of detail. Strategic level changes are to be achieved within the next five to ten years and need to be made more tangible through a more detailed planning on the tactical level. Within the operational level the most detailed planning takes place and considers workpackages and detailed project information in the near term future (up to two years from now). The current architecture is modeled at all three levels, whereas only parts of the target architecture are modeled at strategic, tactical, and operational level.

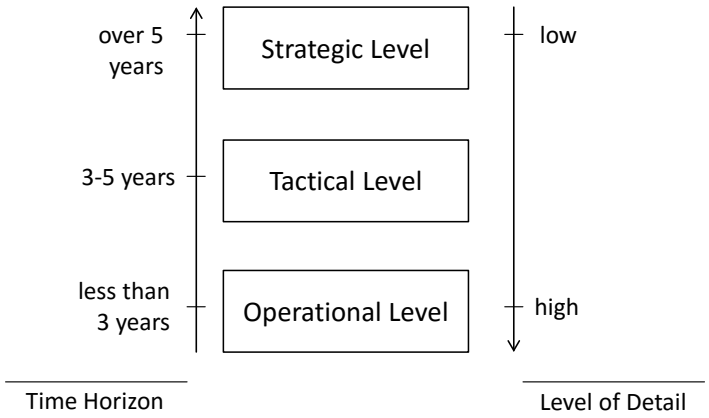

Figure 2.2: Time Horizons and Level of Detail for Different Management Levels

A general EA planning process consists of six steps (Aier et al., 2009). Figure 2.3 shows the steps, their process flow and a feedback flow. At first a vision for the architecture is defined, that serves as a guidance for the development of the models used for planning. In the second step the current architecture is modeled and serves as the planning basis. Afterwards, in the third step alternative target architectures are modeled. In step four these alternatives are analyzed and evaluated. A target architecture is selected to be pursued and in step five the transformation is planned to change the current into the target architecture. The feasibility flow from step five back to step three allows to reintegrate information gathered at step five and with relevance for the models of the target architecture. A replanning might be necessary as the assumptions made at step three might not hold any longer. In step six the transformation is actually implemented. The implementation in this context does not only refer to IT development activities, but also to organizational changes.

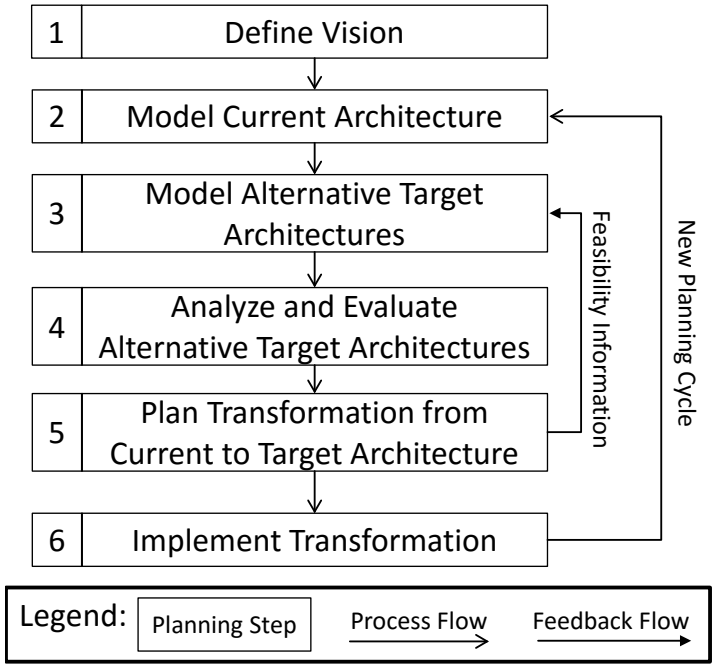

Figure 2.3: Enterprise Architecture Planning Process; Adapted from Aier et al. (2009, Figure 3)

2.1.1 Transformation Model

Hitherto, we introduced just the EA models for different points in time. However, for planning a transformation from a current to a target architecture we need also information that allows us to determine the relations between the elements for different points in time. The transformation model is a model that contains information about the successor relationships of elements for enterprise architecture models at two different points in

time, for example current and target architecture (Aier and Gleichauf, 2010a). For two models at the same point in time no transformation model exists. According to Aier and Gleichauf (2010a) the transformation model is created by comparing two models and identifying the successor relationships. Six different types of successor relationship bundles exist:

- *noSuccessor*
- *noPredecessor*
- *oneToOne*
- *manyToOne*
- *oneToMany*
- *manyToMany*

A bundle of successor relationships is a combination of elements, their successors and successor relationships which allows to differentiate them from all other bundles as the bundles are disjoint. For example an application that has no successor application in the target architecture and is not a successor of itself is considered as a bundle of the type *noSuccessor*.

Given a current architecture, a target architecture and their transformation model it is possible to relate each model element and successor relationship to exactly one bundle. Each of the bundle corresponds to one bundle type. We explain the successor relationship bundles in the following. It is possible that an element has no successor (*noSuccessor*) or that an element has no predecessor (*noPredecessor*). Furthermore, it is possible that an element is replaced by exactly one element or stays the same, and therefore it is its own successor (*oneToOne*). Many elements can be consolidated into one successor (*manyToOne*), but also splitting one element into several successors is possible (*oneToMany*). A complex successor relationship bundle is present when many elements have many successors that converge (*manyToMany*). By refining the transformation model it is possible to detail necessary development activities.

2.1.2 Lifecycle phases of Elements

The Generalised Enterprise Reference Architecture and Methodology (GERAM) (IFIP-IFAC Task Force, 1999) is the foundation for the 15704:2000 standard from the International Organisation for Standardization (International Organization for Standardization, 2000). Instead of defining a concrete enterprise architecture methodology and model, they provide definitions, guidelines, and building blocks that a sound EAM approach

and tool should consider. An emphasis is put on the role of lifecycles which are inherent in every element of the enterprise architecture and allow for the consideration of the life history of elements. Examples for elements are business processes, applications and projects.

The first lifecycle phase every element enters is the identification phase, which is followed by the concept phase. After the conceptual nature of an element has been determined the requirements phase follows that builds the basis for the design phase of the element. Designing an element means to further specify the properties and characteristics of an element to allow for an implementation in the subsequent implementation phase. After an implementation took place the element enters the operation phase in which it is in use in the enterprise. The last phase is the decommission of the element which means that it is retired. An element can enter and exit the same lifecycle phase during its existence several times, as for example an application might be redesigned and reimplemented. Furthermore, the lifecycle phase of an element may trigger the change of other elements' lifecycle phases. The importance of the lifecycle phases for transformation paths is that the transformation paths determine the planned points in time. Within the transformation path elements are developed, stay operational or get retired. Each of these possible situations provide different benefits or drawbacks which have to be considered in transformation path generation. In some situations the prolongation of an application's operational lifecycle phase might be regarded as beneficial whereas in others the development of a new application and its retirement might be regarded as beneficial.

2.1.3 Types of changes in IT Transformations

It is possible to differentiate between three different types of transformations that are applicable to enterprise architecture elements: create, update, and delete. Within the application architecture, as part of the enterprise architecture, applications and application services are the elements that are created, updated, and deleted (Simon, 2009; Postina, 2012; Hanschke, 2013; Keller, 2007). Creating is to introduce a new element to the enterprise architecture. Updating means to modify elements by changing their relationships or attributes. Deleting is to remove elements from the enterprise architecture, respectively to set them to a non-operational lifecycle phase. It is possible that the creation, deletion, and update of different elements interrelate to each other. For example, the update (change) of an application can interrelate with the deletion of one of its provided application services by creating a new one. In this case the application would be a successor of itself and the new application service would be the successor of the one that is going to be retired.

2.1.4 Reference Scenarios for Sequencing Changes

The Quasar Enterprise approach from Engels et al. (2008) assists in the design of service-oriented application landscapes. Application landscapes can be considered as part of an enterprise architecture. For sequencing changes into steps Engels et al. provide reference scenarios. "A reference scenario is a sequence of elementary changes in an application landscape, that has proven its applicability for certain differences between current and target application landscapes" (Engels et al., 2008, p. 292). We consider elementary changes as those that are of importance for an enterprise architect and assume that the mentioned differences are gaps as defined in Section 3.1.

Quasar Enterprise introduces two reference scenarios in detail. The first is the reference scenario "Inventory components first: If the to-be contains an inventory application component which is to be developed or changed, this change should be layed out first." (Engels et al., 2008, p. 292). It consists of four steps and starts with the implementation of the inventory component, that can be considered as an application or part of an application. In the second step a temporary data integration between the inventory component and the application that implements the inventory functionality is established via a temporary data dependency. All applications that use the functionality, that is to be replaced, are already redirected to a temporarily created data application service. In the third step the temporary data application service is replaced through a logic application service. In the fourth and last step the temporary data dependency is removed and reversed by establishing one to the logic application service as specified in the target enterprise architecture (to-be).

The Second reference scenario is the "Migrate the application services of supporting business services before core business services" (Engels et al., 2008, p. 294). It consists of three steps, where the first step is the creation of the supporting application (component). As a second step the supporting functionality that is to be carved out from an application (component), with mixed functionality, is temporarily provided to the newly created application. In the third step this functionality is provided from the newly created application and now used from the application, that is no longer mixed.

2.2 Enterprise Architecture Frameworks

EA Frameworks are a combination of methods and models, which enable an enterprise to introduce, establish, and improve EAM as a management discipline. A multitude of EA frameworks exists that consider different stakeholders, architecture layers, and elements within the layers (Matthes, 2011).

One of the first EA frameworks published was the Framework for Information Systems Architecture, also called Zachman Framework named after its inventor (Zachman, 1987). The framework postulates that it is necessary to provide different models for solving different issues that the people involved in the changes within the enterprise have to cope with. However, these models are interconnected, may it be implicit or explicit, and can be formalized (Sowa and Zachman, 1992). The framework suggests to cope with this interconnectedness in a systematic way to successfully change the enterprise and its IT.

One of the first EA frameworks in Europe is the Architecture of Integrated Information Systems (ARIS) published in Scheer (1991). It is especially well adopted in German-speaking regions and provides a methodology for describing information systems by means of models (Matthes, 2011). ARIS provides different views that allow to model business processes and their relations to the IT. The views consider static and dynamic aspects, resources, and the products created by the enterprise.

According to Matthes (2011) more than fifty frameworks exists. In the following we introduce an internationally accepted and widely adopted framework for enterprise architectures from the Open Group[1]. The Open Group is a global consortium that has approved several IT standards in collaboration with industry partners.

The Open Group Architecture Framework

The framework from The Open Group for EAM is called The Open Group Architecture Framework (TOGAF)(The Open Group, 2011) and consists of seven parts. Part one is an introduction to the standard, its core concepts, and definitions. In Part two the TOGAF Architecture Development Method (ADM) is introduced. Figure 2.4 shows the relations between the phases of the architecture development cycle, as part of the ADM.

The ADM consists of eight sequential phases, one preliminary phase, and one continuous requirements management phase that interacts with all other phases. Using the ADM helps an enterprise to transform itself by creating an architecture vision (phase A), and modeling current and target architectures for business, information systems, and technology (phases B, C, and D). The remaining phases (E to H) are concerned with deciding upon changes, planning the migration from current to target architecture, choosing how the change is going to be implemented and observing how the change progress proceeds.

[1] http://www.opengroup.org/

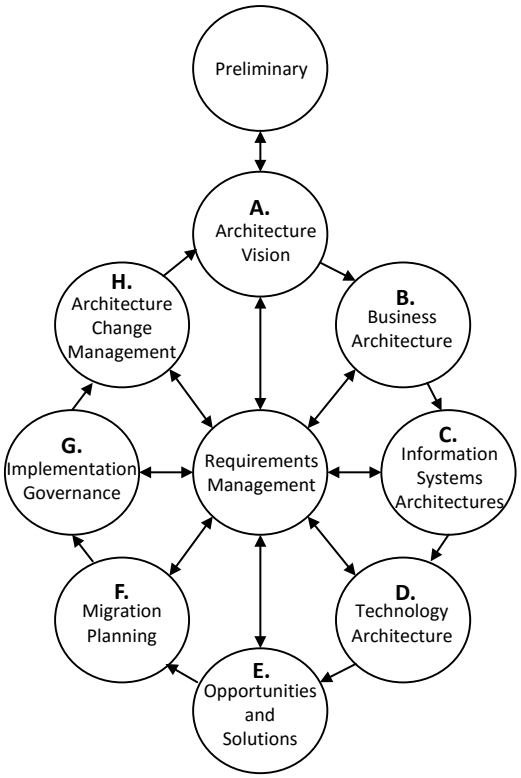

Figure 2.4: TOGAF Architecture Development Cycle; From The Open Group (2011, Figure 5.1)

Part three of the TOGAF standard provides guidelines and techniques which help to use the ADM. Chapter 27 of TOGAF provides an introduction to the gap analysis as a key activity at the end of the phases B, C and D of its ADM. As a consequence gaps relate to the business, information systems, consisting of data and application, and technology domain. As a procedure to identify gaps TOGAF suggests drawing up a matrix with all building blocks of the to-be model on the horizontal axis and all building blocks of the as-is on the vertical axis. This approach has the shortcoming that it only considers added and removed building blocks but not how to cope with changed building blocks, even though impacted building blocks are considered as important. Furthermore, data relationship gaps are considered as an important outcome of the gap analysis, but a description how to derive these gaps is missing.

Part four of TOGAF introduces the content framework, which contains a content meta-model, links the elements to ADM phases, and introduces the concept of building blocks. Building blocks are recognizable something that is of interest for a stakeholder and provide a package of functionality. However, a building block should be decoupled from its implementation and thus allow for its reuse in different contexts. The content meta-model considers five architecture domains: business (1), data (2), application (3), technology (4), and architecture principles, requirements, and roadmap (5).

Six extensions are already provided off the shelf to satisfy information demands ranging from modeling business process details or modeling a service oriented architecture, over to modeling data details comparable to an entity relationship diagram.

In part five the concepts of enterprise continuum, architecture partitioning, architecture repository, and tools for architecture development are presented. The continuum allows an enterprise to refine the content meta-model that fits its organization specific requirements, and by partitioning it can build segments of its architecture to improve the manageability of the EA. An architecture repository contains the meta-model, the models of the architectures itself, and further information regarding architecture standards, guidelines, and governance of the EA. This information, the meta-model and models should be made accessible via a tool to provide appropriate views and change functionality.

Part six introduces two reference models which help to classify services and the implementing software of the technology and application architecture according to taxonomies. The technical reference model (TRM) provides a taxonomy for the application platform services, which can be regarded as middleware services. In contrast the integrated information infrastructure reference model (III-RM) is concerned with the exchange of information between applications for satisfying user informations needs. The III-RM also provides a taxonomy which is a refinement of the TRM.

In TOGAF's part seven an architecture capability framework is presented, that allows an enterprise to establish a management of its enterprise architecture. Besides, guidelines on how the EAM supports in which phase of the ADM, further hints are provided on how to measure the maturity of the EAM. The latter is important to allow for an incremental introduction of the EAM, and its variety of support, into an enterprise. Furthermore, part seven introduces the concept of an architecture board and its role and responsibilities within the organization of the enterprise.

2.3 Knowledge Management and Semantic Web Technologies

Even though knowledge management and the semantic web have common domains they are used in, they have some differences that justify the existence of each concept on its own. We describe the key points for both concepts in the following Subsections to clarify the differences between them. Furthermore, we will reuse both concepts later on in the thesis.

2.3.1 Knowledge Management

Knowledge management is concerned with storing, retrieving and deploying knowledge (French et al., 2009). Computer systems that support the management of domain knowledge are called knowledge-based systems (Russell and Norvig, 2010) or expert systems (French et al., 2009). These systems mainly consist of a knowledge base, an inference engine, and a user interface. A knowledge base contains or has links to all the data that is available from different data sources. Based on this data an inference engine is able to derive implicit data in the knowledge base. Furthermore, the inference machine is able to detect inconsistencies in the knowledge base. However, the inference machine needs, besides the data from the knowledge base, formalized knowledge of the domain, that allows the inference engine to detect an inconsistency and to reason new information. A user interface allows to represent the information in a visual way to the user, which allows her to grasp the information.

If the domain, that is to be formalized, is enterprise architecture and a knowledge-based system allows to manage the models, then the architecture repository is the knowledge base (Chen et al., 2013).

2.3.2 Semantic Web Technologies

Semantic web technologies are used to integrate heterogeneous data sets and formalize the underlying structure of the information to allow a machine to understand the semantics of it (Shadbolt et al., 2006). The World Wide Web Consortium (W3C) provides a set of standards to describe an ontology and to query it. An ontology "is a set of precise descriptive statements about some part of the world (usually referred to as the domain of interest or the subject matter of the ontology)" (Motik et al., 2009). Besides the ability to reuse existing data, a further advantage of semantic web technologies is the flexibility of the ontologies to allow for an evolution of the representation of the domain of interest.

Two standards are of relevance for a proposed technical realization: firstly, the Web Ontology Language (OWL 2) (Motik et al., 2009) for making descriptive statements and secondly, the SPARQL Query Language for RDF (SPARQL) (Prud'hommeaux and Seaborne, 2008), which allows querying these statements.

The Resource Description Framework (RDF) (Manola et al., 2004) is a basis for both standards, as OWL ontologies can be serialized as RDF graphs and can be accessed via SPARQL. An RDF graph consists of triples of the form 'subject, predicate, object', where subjects and objects are nodes and predicates are relations. Every resource in an ontology is identified by a resource identifier which allows for example distinguishing

between a bank in a financial context and a riverbank. Information from the ontology is queried via SPARQL, which provides the resources that match patterns specified within the query.

Semantic web technologies have already been applied to domains of interest that range from semantic business process modeling (Lautenbacher, 2010) to diagnosis of embedded systems (Grimm et al., 2012). First implementations based upon semantic web technologies for EAM already exist from TopQuadrant with its TopBraid Composer[2] and Essential Project[3].

Chen et al. (2013) present an approach to facilitate reuse of existing information within an enterprise architecture repository. The approach consists of four steps and starts with the formalization of the data sources that contain data which is of interest for the enterprise architect. In the second step the formalization is abstracted to the OWL 2 QL profile which guarantees a query answering in LOGSPACE regarding the size of the data set. Linking the different data sets is done in the third step by providing mappings between concepts. In the fourth and last step the mappings can be refined manually. If formalizations of the data sets are already available in OWL 2 they can be reused. This is for example the case for the ontology of the TOGAF content metamodel (Gerber et al., 2010) and the Business Process Modeling and Notation (BPMN) (Natschläger, 2011).

2.4 Formalisms for Graph Grammars

In the following we introduce graph grammars and their underlying formalisms that we will reuse later in the main chapters of the thesis. Graphs and graph transformations in general have the benefit that they have a sound theoretical foundation (Rozenberg, 1997). However, they provide different degrees of expressiveness that include benefits and drawbacks regarding information that can be modeled. Furthermore, the infinity of the graphs that are generated through graph transformations needs to be considered when using graphs as a formalism.

2.4.1 E-Graph and E-Graph Morphisms

We start with defining E-graphs as an extension of graphs to allow for a consideration of attributes for nodes and edges within a graph. A graph in general consists of nodes (vertices) and edges, whereas the edges connect nodes and represent the relationships

[2] www.topquadrant.com/docs/whitepapers/WP-BuildingSemanticEASolutions-withTopBraid.pdf
[3] www.enterprise-architecture.org/

between entities. Edges and nodes may be labelled or typed to give graphs a meaning to enhance their pragmatics. Labelling, in contrast to typing, is not an explicit formalism for defining the metamodel of graphs (Rensink, 2010, p. 14).

We introduce E-graphs as a graph formalism and reuse the definition from Ehrig et al. (2006, p. 172):

Definition 1. *An E-graph G is defined as*

$G = (V_G, V_D, E_G, E_{NA}, E_{EA}, (source_j, target_j)_{j \in |G,NA,EA|})$ *with:*

- V_G and V_D as finite sets of graph nodes and data nodes
- E_G, E_{NA},E_{EA} as finite sets of edges consisting of graph edges (E_G), node attribute edges (E_{NA}), and edge attribute edges (E_{EA})
- $source_G : E_G \to V_G$, and $target_G : E_G \to V_G$ as source and target function for graph edges
- $source_{NA} : E_{NA} \to V_G$, and $target_{NA} : E_{NA} \to V_D$ as source and target function for node attribute edges
- $source_{EA} : E_{EA} \to E_G$, and $target_{EA} : E_{EA} \to V_D$ as source and target function for edge attribute edges

Figure 2.5 shows how the different nodes and edges are interrelated via the different source and target functions, as defined in Definition 1.

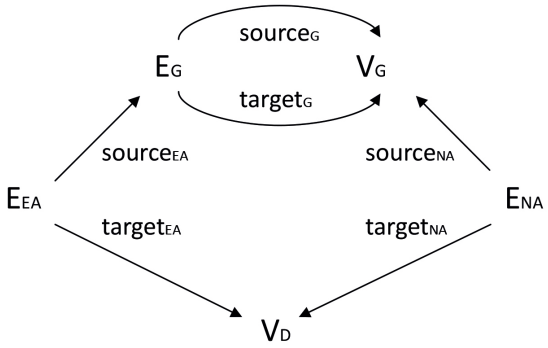

Figure 2.5: Relationships Between the Different Concepts of E-Graphs (Figure from Ehrig et al. (2006, p. 172))

Given two E-graphs G^1 and G^2 with $G^k = (V_g^k, V_D^k, E_G^k, E_{NA}^k, E_{EA}^k, (source_j^k, target_j^k))$ for $k = 1, 2$, we define the necessary morphism for them. This morphism is called E-graph morphism.

Definition 2. *An E-graph morphism $f : G^1 \to G^2$ is defined, according to Ehrig et al. (2006, p. 172), as a tuple $(f_{V_G}, f_{V_D}, f_{E_G}, f_{E_{NA}}, f_{E_{EA}})$ where $f_{V_i} : V_i^1 \to V_i^2$, and $f_{E_j} : E_j^1 \to E_j^2$ with $i \in G, D$ and $j \in G, NA, EA$. The function f preserves the source and target functions:*

- $f_{V_{G,NA,EA}} \circ source^1_{G,NA,EA} = source^2_{G,NA,EA} \circ f_{E_{G,NA,EA}}$
- $f_{V_{G,NA,EA}} \circ target^1_{G,NA,EA} = target^2_{G,NA,EA} \circ f_{E_{G,NA,EA}}$

Graph morphisms in general can have different properties (Boneva et al., 2007). A graph morphism is total if f_V and f_E are total functions. In contrast a partial graph morphism is present if f_V and f_E are total functions from a G' to H, where G' is a subgraph of G. A graph morphism is injective if for every element in H exists only one or none matching from G. Bijective graph morphisms require such a match so that every element in H has a match in G. A bijective graph morphism is also called isomorphism (Rensink and Zambon, 2012, p. 67).

2.4.2 Attributed Graph and Corresponding Morphism

E-graphs can be extended by a data algebra D to specify a carrier set D_S for the data nodes V_D (Brandt, 2013, p. 120). OP_D are operations to manipulate the data values.

Definition 3. *Ehrig et al. (2006, p. 173) define an attributed graph $AG = (G, D)$ as an E-graph G and a $DSIG$-algebra D, where $DSIG = (S_D, OP_D)$.*

The corresponding graph morphism for the attributed graphs needs to be extended accordingly. Therefore, we define:

Definition 4. *An attributed graph morphism $f : AG^1 \to AG^2$ for two attributed graphs AG^1, AG^2 is a pair $f = (f_G, f_D)$, with an E-graph morphism f_G, and an algebra homomorphism $f_D : D^1 \to D^2$ (Ehrig et al., 2006, p. 173).*

2.4.3 Attributed Type Graph and Typed Attributed Graph

In the following we introduce attributed type graphs as a formalism for specifying a metamodel for graphs.

Definition 5. *An attributed type graph $ATG = (TG, Z)$, according to Ehrig et al. (2006, p. 175), consists of:*

- a type graph TG
- a final $DSIG$-algebra Z

Graphs that correspond to an attributed type graph are called typed attributed graphs. They can be considered as valid models for a specified metamodel. For the definition of a typed attributed graph we reuse the definition from Ehrig et al. (2006, p. 175):

Definition 6. *A typed attributed graph $TAG = (AG, t)$ consists of*

- an attributed graph AG
- an attributed graph morphism $t : AG \to ATG$, where ATG is an attributed type graph

Finally, we need to adapt the graph morphism to consider types and attributes. Therefore, we define, in accordance with Ehrig et al. (2006, p. 175):

Definition 7. *A typed attributed graph morphism $f : (AG^1, t^1) \to (AG^2, t^2)$ is an attributed graph morphism with:*

- $f : AG^1 \to AG^2$
- and $t^2 \circ f = t^1$

2.4.4 Typed Attributed Graph Productions and Transformations

So far we defined only the structures, expressiveness and typing of graphs. However, we want to use the formalisms also to specify changes on the graphs. Therefore, we introduce typed attributed graph productions. They allow us to specify changes to typed attributed graphs.

Definition 8. *A typed attributed graph production is according to Ehrig et al. (2006, p. 182) defined as $p = LHS \xleftarrow{l} K \xrightarrow{r} RHS$ with:*

- LHS, RHS, and K as typed attributed graphs with a shared $DSIG$-termalgebra $T_{DISG}(X)$
- LHS as the left hand side of a typed attributed graph production
- RHS as the right hand side of a typed attributed graph production

- K as an interface graph between LHS and RHS
- $l : K \to LHS$ as an injectively typed attributed graph morphism
- $r : K \to RHS$ as an injectively typed attributed graph morphism

However, specifying changes with typed attributed graph productions needs to be instantiated in an typed attributed graph TAG to apply actual changes. The actual transformation of a graph with a typed attributed graph production p is a typed attributed graph transformation. Therefore, we define:

Definition 9. *A direct typed attributed graph transformation is defined as* $TAG_G \overset{p,m}{\Longrightarrow} TAG_H$*, according to Ehrig et al. (2006, p. 182):*

- TAG_G and TAG_H as typed attributed graphs
- $m : LHS \to ATG_G$ as a typed attributed graph morphism (match)
- p as a typed attributed graph production

2.4.5 Single and Double Pushout Approach

In general two algebraic approaches to graph transformations exist: the single and the double pushout approach. The double pushout approach needs two matches via an interface graph K as a gluing condition for applying graph productions. The gluing condition consists of a dangling edge condition and an identification condition (Ehrig et al., 2006, p. 188). Gluing points GP are elements of p that are part of LHS and RHS. This means that they do not change through the production. Identification points IP are determined by $m : LHS \to TAG_G$ and dangling edge points DP are those nodes of TAG_G that are to be deleted by the application of p. If $IP \cup DP \subseteq GP$ the gluing condition is satisfied and p is applicable (Ehrig et al., 2006, p. 188). If one of both or both conditions are not satisfied the interface graph K cannot be constructed and as the consequence the typed attributed graph transformation is not applicable.

In contrast the single pushout approach for transformations just needs a match of LHS in TAG_G. As a consequence dangling edges can be deleted through the application of a graph transformation and prohibit the creation of an inverse typed attributed graph transformation. Besides the dangling edge condition the single pushout approach has the drawback of neglecting identification condition. However, ensuring the dangling edge condition and identification condition can be realized in single pushout approaches via negative application conditions (Habel et al., 1996).

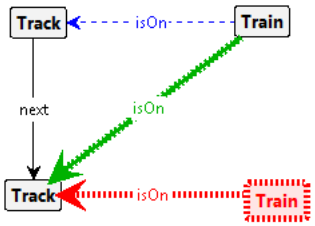

Figure 2.6: Example of a Graph Production MoveTrain in the Railroad Domain with a NAC

A negative application conditions (NAC) allows to specify constraints on the LHS by stating which nodes and edges are forbidden for a typed attributed graph production to match (Ehrig et al., 1997). Furthermore, a NAC can specify that two nodes are to be non-identical for a typed attributed graph production to match. This type of constraints is especially important for the single pushout approach to prohibit an injective matching to the same node in the left hand side.

Figure 2.6 shows the example of a graph production that moves a train from one track to another with the NAC that there is not already a train on that track. We decided to use an example from the railroad domain, as many practical examples in publications also refer to them (Habel and Pennemann, 2005; Estler and Wehrheim, 2011) and we consider the domain as one that is easy to digest. Nevertheless, the application scenarios have a huge variety from ant colony simulation to the transformation from one business process language into another (Ghamarian et al., 2012). The blue dashed edge, in Figure 2.6, signalizes that this edge is only part of the right hand side and is deleted through the application of the graph transformation. In contrast the green edge reflects the fact that the train was not on this track (edge is not part of the left hand side), but is present after the application of the graph transformation (edge is part of the right hand side). The black edges and nodes in Figure 2.6 are the parts of the graph that remain unchanged. They are part of the LHS and the RHS of the typed attributed graph production.

Nested application conditions are a more general concept of graph constraints and application conditions (Ehrig et al., 2006, p. 71). They allow for expressing the $\exists\neg(\exists)^n$ fragment of first-order logic, where n is the height of a stack of nested application conditions of level n (Rensink, 2004, p. 321). Each level of nesting corresponds to a graph sub-production that needs a morphism in the graph to be applicable. For the proofs regarding the correctness and generality of the formalism we refer to Habel and Pennemann (2005) and Rensink (2004). Figure 2.7 shows the example of a graph production that loads all cargo into a train for all the tracks the train is on. The figure encodes the following formulae:

$\exists t : Train(t) \wedge (\forall y, z : Track(y) \wedge Cargo(z) \wedge isOn(t, y) \wedge isAt(z, y)) \Rightarrow \forall t, y, z : hasLoad(t, z) \wedge \neg isAt(y, z)$

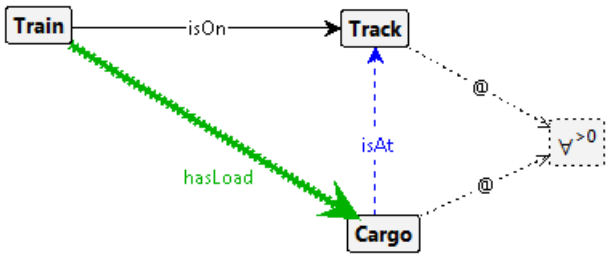

Figure 2.7: Example of a Nested Application Condition in the Railroad Domain for a Graph Production LoadCargo

Please note that in this case a train can not only be on one track. Such a limitation can be enforced through a type graph. LoadCargo would be applicable anyway as it does match already if the train is on one track.

2.4.6 Typed Graph Transformation System and Graph Grammar

Now that, we have defined the graphs and their changes we introduce typed graph transformation systems as a state-space representation system. Each graph can be considered as a state that is changed through the application of a typed attributed graph transformation, whereas the transformation corresponds to a state transition. We reuse the definition from Ehrig et al. (2006, p. 183):

Definition 10. *A typed graph transformation system is defined as GTS = (DSIG,ATG, P) with:*

- $DSIG$ as a data type signature
- ATG as an attributed type graph
- P as a set of typed attributed graph productions

However, the actual changes using a graph transformation system and the corresponding state-space only become explorable when they are instantiable on a typed attributed graph that serves as the initial state. The formalism that covers this instantiation is called graph grammar.

Definition 11. *A typed graph grammar is defined as* $GG = (GTS, TAG_i)$ *with (Ehrig et al., 2006, p. 183):*

- GTS is a typed attributed graph transformation system

- TAG_i is typed attributed initial graph of the grammar

A language L generated by a typed graph grammar is given through $L = \{G|S \Rightarrow^* G\}$, where $\Rightarrow^*$ is a sequence of direct graph transformations $TAG_{G_0} \Rightarrow TAG_{G_1} \Rightarrow ... \Rightarrow TAG_{G_n}$ transforming the initial graph into other typed attributed graphs or itself (Ehrig et al., 2006, p. 183). The latter case is present for cyclic typed graph transformation systems.

2.5 Automated Planning

Several different approaches, techniques and representations to planning problems have been developed over the last decades (Russell and Norvig, 2010; Ghallab et al., 2004). These approaches range from state space model based planning to task networks, where tasks for reaching a goal are decomposed and sequenced. Automated planners solve planning problems by using a set of actions to create a correct solution that satisfies a goal. Figure 2.8 shows the different concepts used in automated planning and their relations. We introduce the concepts below.

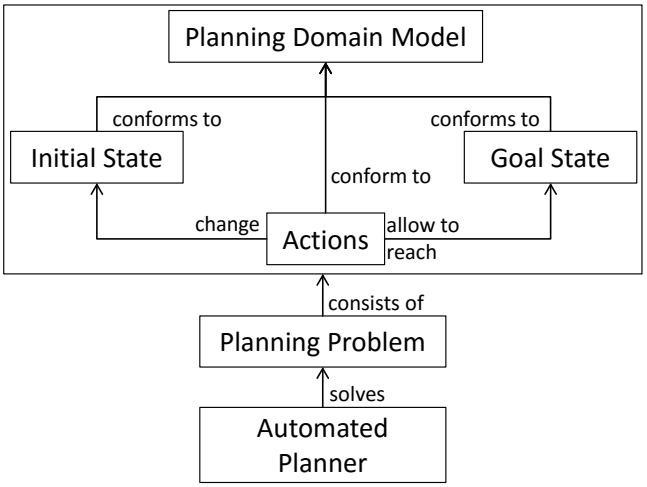

Figure 2.8: Automated Planning Concepts and Their Relations

Every automated planner needs some information on the state of the world. This information is based on a planning domain model which contains all the metamodel information of the planning domain. An initial state is created by building an instance model of the state of the world that is conform to the planning domain model. With the initial state the automated planner starts to generate a plan that achieves the goal. The automated planner generates a plan by computing applicable actions and the consequences an execution would have. A goal can be composed of several other goals. Furthermore, goals can make statements about properties of a state or a set of actions has to satisfy.

Furthermore, it is possible to explicit state relationships, attributes, and elements that have to hold in a state to be a goal state. A combination of properties and explicit statements for goal definition is also possible.

Actions are commonly described on elements, relationships, and attributes of a planning domain model, as this allows their multi-instantiation in the plan generation. The same action may be applicable on two different elements and thus be instantiated multiple times. An action which is coupled to a specific element's identity lacks such a generality.

As the number of states or set of applicable actions is possibly infinite and the automated planner does not always know if the currently explored states and actions help to solve the planning problem heuristics and search strategies are used to guide its search. A search strategy defines the way in which the space of possible solutions is explored, whereas a heuristic determines a value for the explored (partial) solutions to guide the further exploration. Heuristics help to narrow the set of all further explorable information to the set of the most promising in respect to the goal. Depending on the planning domain and goal different strategies and heuristics are more or less supportive.

2.5.1 Inference Engine as an Automated Planner

An inference engine, using axiomatic inference, could be used to solve planning problems (Ghallab et al., 2004, p. 39). The three major inference algorithms, in first-order logic, that allow an inference engine to draw conclusions are forward chaining, backward chaining and theorem proving (Russell and Norvig, 2010, p. 322). They are well suited for deriving implicit knowledge and detecting inconsistencies. However, reasoning such implicit knowledge is different from applying changes to a model and not well suited for solving planning problems, as we want to create plans in terms of incremental changes to an initial state.

Furthermore, from a pragmatic point of view we state, that we want the knowledge base to be as is, and not influenced by the possible future states. Therefore, we discard the inference engine as an automated planner.

2.5.2 Graph Transformation Based Planning Formalism

In the following we describe our rationale for using graph transformations as a formalism to create transformation paths supported by an automated planner. A state space based approach is preferable, because models of the current and target architecture are used in many EAM frameworks (The Open Group, 2011, 2012; Niemann, 2006; Hanschke, 2009)

and are present in tools used in practice (Matthes et al., 2008). Furthermore, we want to allow enterprise architects, as domain experts, to interactively create the transformation paths with an automated planner. As a consequence intermediate states are of relevance in the generation of the transformation path. Therefore, plan-space planning and propositional satisfiability techniques are discarded, as they do not support the creation of such information. The last remaining planning approach to be considered are planning-graph techniques Ghallab et al. (2004, Chapter 6). These techniques establish a sequence on sets of actions. However, the way in which plans are generated are not suitable for both exploring different target architectures and generating sequences of actions. Therefore, we discard planning-graph techniques, too.

The de facto standard in the scientific community dealing with automated planning is the Planning Domain Definition Language (PDDL) (Edelkamp and Hoffmann, 2004). It is a planning domain independent language that allows to formalize states, actions, and goals and is interpretable for many different implementations of automated planners. PDDL allows to describe planning problems to be able to benchmark different planning approaches and algorithms. However, PDDL has two shortcomings to be used in our domain. Firstly, one cannot express goal states with quantifiers in terms of 'for all' elements of a certain type must hold a specific property. Secondly, it is not possible to specify constraints on the states constructed during planning (Russell and Norvig, 2010, p. 388) and the construction and deletion of objects during the planning process is not possible (Estler and Wehrheim, 2011). Therefore, we use a graph transformation system based planning approach.

2.5.3 Graph Transformation Systems for Automated Planning

Graph transformations for automated planning solve a planning problem by applying graph transformations to a model until a solution for the planning problem is found. The graph transformations are the actions that span a graph transformation system. Each node is a state and each transition is an edge from one state to another state. Actions with no consequence for the state are self-directed transitions as the source and target of the transition are the same. Each action is specified as graph transformation using a left hand side, a right hand side and negative application condition. The definition of a goal state can be explicit for the whole state, or only parts of it, or even defined in terms of properties a goal state has to satisfy. Independent from the degree of explicitness the goal is encoded in a special graph transformation which allows to identify goal states and mark them (c.f. Estler and Wehrheim (2011)).

However, graph transformations have the disadvantage that they provide a huge state space regarding the states, which have to be examined when all states in the graph are computed. As a consequence this influences the computation time of all possible worlds created through the transformations. The benefit of graph transformations is their

capability to distinguish between situations, i. e. the different transitions that lead into the same state. Furthermore, the usage of graph patterns allows to specify changes to the models on different levels of abstraction, that also enhance the specification of goal states. With graph transformations a planning problem can be solved by searching for graph patterns in a state represented by a graph and applying graph transformations to change the state (Edelkamp and Rensink, 2007).

2.6 Multi-criteria Decision Making

Decision making is the act of intentionally taking a choice, out of a set of decisions, that has consequences on the present and the future. The research area that examines the decision theory behind is called multi-criteria decision making (MCDM) for discrete decision spaces, respectively multi-objective decision making (MODM) for continuous decision spaces (Triantaphyllou, 2000, p. 1). Discrete decision space means that the number of choices of decisions is predetermined and solution cannot shift gradually. Each discrete choice is an alternative decision which consists of multiple attributes which determine the criteria which allow to derive the value of a decision. Those criteria are weighed to be able to express preferences of a decision maker between the different criteria.

MCDM takes also different units into account, that might be used to express the amount for each criteria. Each decision problem can be represented as a decision table which allows to express the value of each alternative for a certain criterion. A MCDM technique is a method to transform the decision table into an overall ranking, that allows the decision maker to derive the best decision. There are also cases in which not only the best alternative is of interest, but instead a certain number of decisions is of interest. MCDM techniques which allow to compare criteria of different units are called dimensionless, as they allow to calculate an overall utility value for a decision alternative aggregated from the criteria.

A sensitivity analysis is used to check the robustness of the ranking for a given set of alternative decisions, their weighted criteria and the MCDM technique used to create the ranking. The sensitivity analysis allows to derive the minimal changes in the weightings and the values of criteria necessary to change the ranking of the best decision or any two decisions. The phenomenon called rank reversal is present in a MCDM problem if for a given ranking alternative A1 is better than A2 and after adding a further criterion, where A1 and A2 are equal, however A2 gets better ranked than A1, i.e. they reverse their rank.

MCDM techniques can either require no input from the decision maker or information regarding the criteria and their weighting. Those techniques that require input are either

expecting standard level, or ordinal, or cardinal input (Triantaphyllou, 2000). Using standard level input means to establish for all criteria of the alternatives predefined values to allow for a ranking using conjunctive and disjunctive methods. With ordinal input, expressing the ordering between the different criteria, lexicographic methods can be applied for ranking alternatives. Cardinal input allows to assign arbitrary values to the criteria, but requires their normalization and their comparability needs to be assured.

In general a process for decisions making consists of the following activities:

1. Identify decision alternatives and criteria to be considered in the ranking
2. Weight the criteria and enter the performance in the decision table
3. Process the decision table and calculate the ranking

In the following we present and discuss three different MCDM techniques that require cardinal input. For an overview of several more techniques we refer to Triantaphyllou (2000).

2.6.1 Weighted Sum Model

The Weighted Sum Model (WSM) is a MCDM technique that follows the additive utility assumption. This assumption states that the benefit of every alternative is determined by the sum of its added values. The WSM was introduced for the first time by Fishburn (1967). Using the following formula allows to determine the best alternative A^* with the WSM:

$$A^* = max \sum_{j=1}^{n} a_{ij} w_j \text{ for } i = 1, 2, 3, ..., m$$

m defines the number of alternatives and n the number of criteria to be considered within the ranking. Weighting each criteria is done with w_j and a_{ij} determines the value for a criterion (j) of a certain alternative (i). However, the WSM cannot be applied to multi-dimensional MCDM problems. Therefore, we discard the WSM technique for our purposes.

2.6.2 Analytic Hierarchy Process

The Analytic Hierarchy Process (AHP) decomposes a MCDM problem into a system of hierarchies and was first introduced by Saaty (1980). The hierarchies allow to structure the decision problems criteria and their sub-criteria in a way that is easier graspable for decision makers. Figure 2.9 shows two hierarchy levels and decision alternatives for the decision problem of selecting a software component. A detailed description of adequate MCDM techniques for this decision problem is provided by Jadhav and Sonar (2009b).

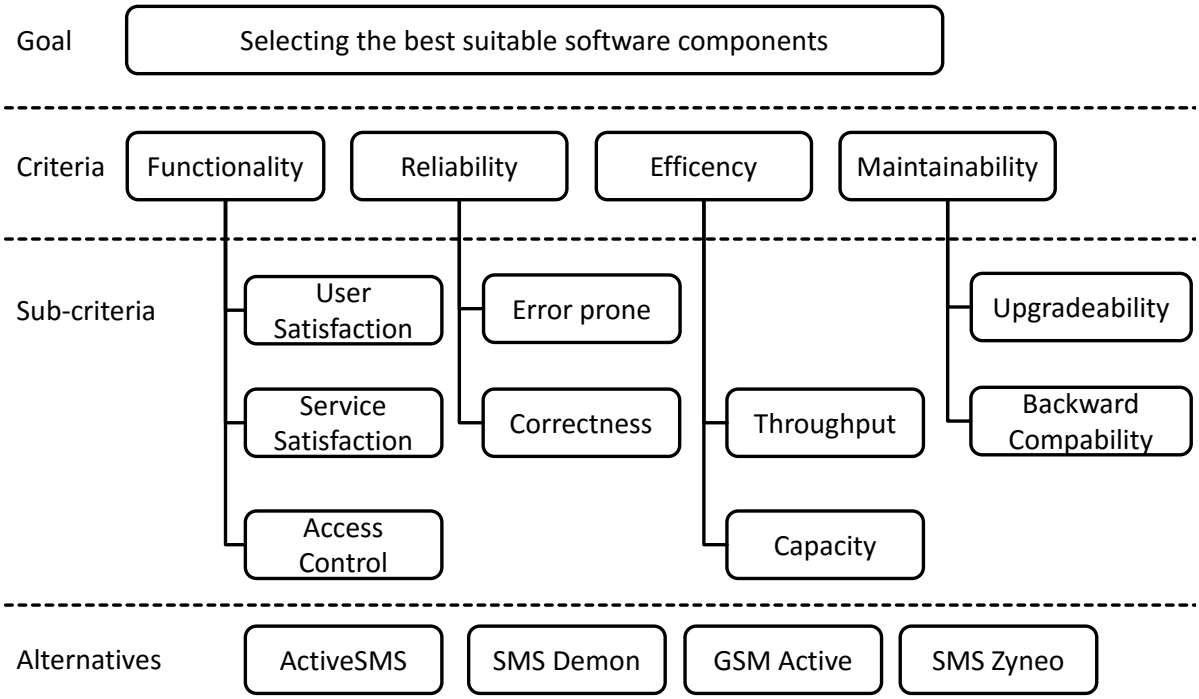

Figure 2.9: Decision Hierarchy and Decision Alternatives for Selecting Software Components; From Jadhav and Sonar (2009a, Figure 4)

However, the decision maker has to make $(\frac{m*(m-1)}{2})$ pairwise comparisons for every criterion between m alternatives (Jadhav and Sonar, 2009a). Furthermore, it is not possible to automatically create a ranking without the input of a decision maker on their performance values for each criterion of each alternative. Therefore, we discard the AHP technique for ranking changes in our approach.

2.6.3 Weighted Product Model

The Weighted Product Model (WPM) is a dimensionless MCDM technique that allows to build a ranking for a set of alternatives. A dimensionless technique ranking is preferable because the factors that play a role in transformation decisions may have incomparable units. Calculating the value of an alternative is done with the following formula:

$$R(A_K) = \prod_{j=1}^{n} (a_{Kj})^{w_j}$$

$R(A_K)$ is the performance value of the alternative A_K which is a product of all weighted criteria $a_{K[1-n]}$ for the alternative. The weighting w_j of the factors is considered as an exponential factor, where the sum of all weights needs to be exactly one. For details on the usage of the WPM we refer to the Subsubsections 4.2.2.2 and 4.3.2.2 where we use the WPM for creating a ranking on transformation actions.

2.7 Methodologies

In the following we present a design process for planning domain models that we followed to create the planning domain model introduced in Section 3.5. Furthermore, we present concepts of a methodology that allows us to demonstrate our approach in Section 7.1.

2.7.1 Design Process for Planning Domain Models

A planning domain model is a metamodel that is used for automated planning and it is designed in an incremental and iterative process (Vaquero et al., 2011, p. 8). Figure 2.10 depicts the phases of the process leaving out arrows for iterations and increments, as it is possible to reiterate from every phase to any earlier phase. After a requirements

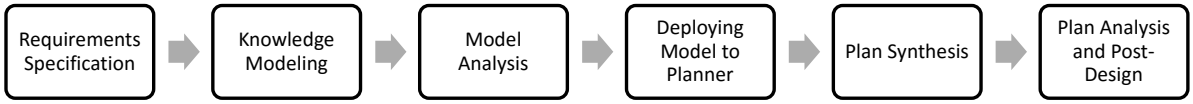

Figure 2.10: Design Process for Planning Domain Models; From Vaquero et al. (2011)

specification of information to be considered in a planning problem a phase for modeling the knowledge follows. Afterwards, the models should be analysed. The analysis includes a verification and validation of the domain model and the planning problem. In the

next phase the model is deployed to a planner which allows to create solutions to a given planning problem. The phase that creates a plan is called plan synthesis. A plan analysis and post-design phase can take place to analyse the created plans and improve potential shortcomings through e.g. creating appropriate heuristics allowing the automated planner to reduce the number of explored states.

2.7.2 Concepts of a Methodology

According to Gutzwiller (1994) a methodology consists of several different concepts that allow for a sound construction and evolution of it. We will use this concepts as a frame to establish a methodology for a so called transformation action repository to demonstrate the feasibility and instantiatiability of our approach (see Section 7.1). We will briefly present the concepts of a methodology in the following:

- **Activities** are necessary tasks to produce the desired output of the methodology. There are one or more start activities followed by several alternative activities that may be conducted only in certain circumstances. One or more end activities together with the other activities are the process model of the methodology.

- **Roles** are the abstract concepts of persons that perform the activities. One person may have several roles within the methodology, but also more than one person may act in the same role as others. The roles are mainly established to determine responsibilities for the activities, but also to be informed within certain activities.

- **Techniques** allow for a detailed guidance of the roles on how to conduct the activities and therefore create the results in the focus of the methodology.

- **Tools** support the application of the techniques and may be implemented in IT.

- **Results** are the outcome of the activities and they are documented in the tools.

- A **Metamodel** constrains the content of the results and determines the conceptual data model for the contents. This allows to document the results and to support the activities and their documentation with IT.

3 Transformation Patterns and Planning Domain Model

In this chapter we introduce a common terminology for the scope of our approach. We proceed with a formalization of the concepts and introduce transformation patterns which allow for an abstraction of modeled situations to their commonalities. We restrict the potentially available transformation patterns to a reasonable and relevant subset. The restriction is established by introducing a feedback loop viewpoint on transformation paths and identifying concepts which are of interest for decision making in transformation path generation. This subset of transformation patterns is then considered within the planning domain model, which is an EA metamodel and the basis for the formalization of transformation actions in the subsequent chapters. Furthermore, we present a mapping of existing EA metamodels to the planning domain model to provide a means for an integration of different models to facilitate a reuse of existing knowledge base information.

3.1 Introduction and Understanding of EA Models for Planning

An EA metamodel defines the elements, relationships, and attributes which can be part of an EA model. Even though, there exists a multitude of EA metamodels (Matthes, 2011) with different terminology (Schönherr, 2009), they share some commonalities which we show in the following. Based on these commonalities we present the concept of transformation patterns that are narrowed down within the planning domain model.

We define an EA model for a certain point in time as an explicit state that contains elements, relationships, and attributes that are present at that point in time. This is in accordance with the maturity of the EA frameworks available (c.f. Buckl and Schweda (2011) and Aier et al. (2009)). Furthermore, it conforms to an EA metamodel and can be outdated if changes are not documented. An EA model can be the composition of several other models such as business process models, service models, and IT infrastruc-

ture models. However, it can also be modeled manually or a combination of manual and composed models (Chen et al., 2013). Transformation planning is concerned with planning changes with the means of EA models. Changing an EA metamodel is not a part of transformation planning activities and thus, the EA metamodel is the same for the EA models used in planning.

We define the current architecture as the model of the current state of the enterprise. Planned or retired elements, relationships, and attributes are not part of the current architecture. There exists only one current architecture for every point in time. We define the target architecture as the model of a desired future state that the enterprise wants to achieve for whatever reasons within a certain time frame. It is possible that alternative target architectures are modeled and compared for planning purposes. A target architecture contains the elements, relationships and attributes that are planned to exist at a point in time in the future. All elements, relationships, and attributes that are not part of the target architecture are planned as to be removed, if not explicitly stated differently. A typical time frame is three to five years (Hanschke, 2013, p. 614).

Given the definition of current and target architecture we now introduce the concept of gaps. Given two EA models we define a gap as a model fragment which is only present in one EA model, but not the other EA model. The type of the gap is either an element gap or a relationship gap or an attribute gap. Figure 3.1 shows the types of gaps. A gap analysis between two EA models derives all gaps between them. Besides the types of gaps it is also possible to determine where the gap has its origin. Either it belongs only to the current architecture and not to the target architecture or vice versa.

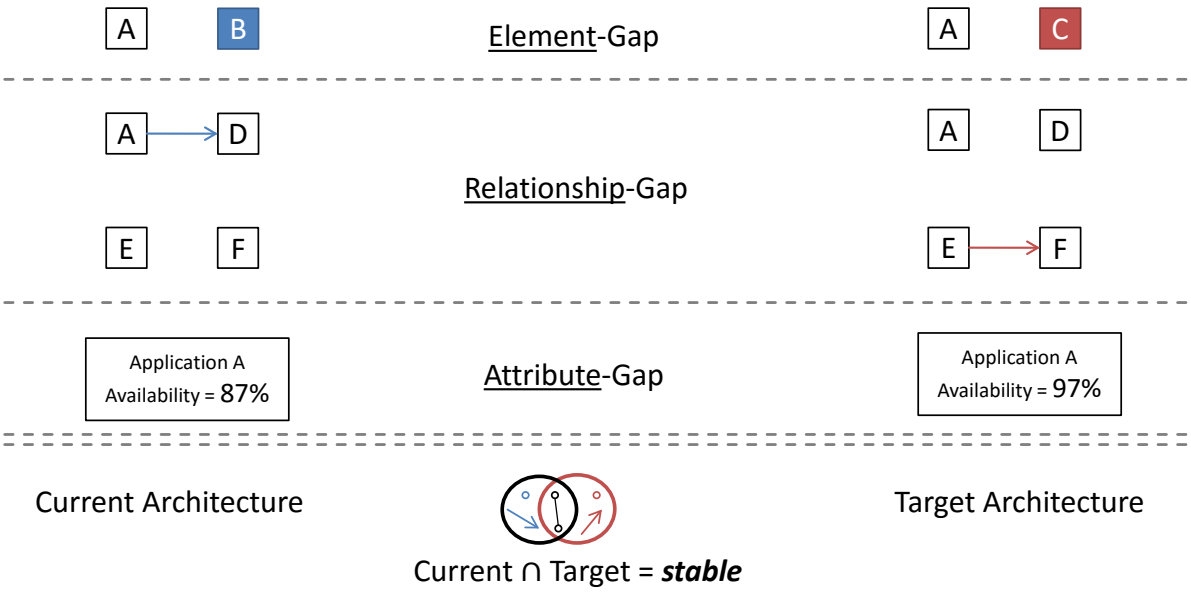

Figure 3.1: Different Types of Gaps

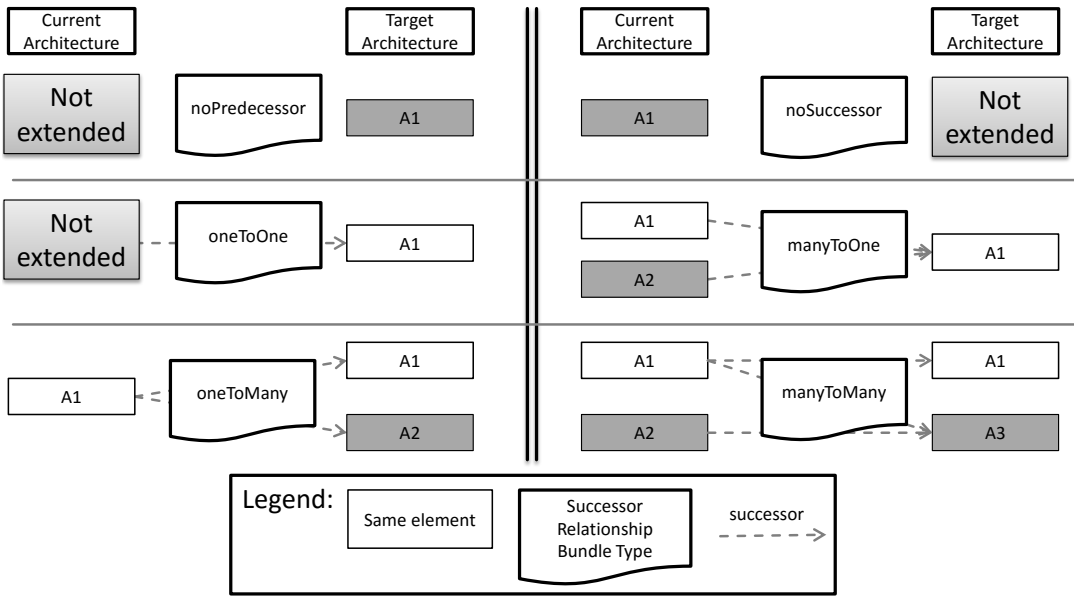

Figure 3.2: Extended Successor Relationship Bundles

As an important part of transformation planning the transformation model has been introduced by Aier and Gleichauf (2010a). We define a transformation model as the model that contains all successor relationships for elements between two EA models for different points in time. From a practical point of view successor relationships for relationships between elements and attributes is not of importance. This information could be considered in the transformation model through a reification of relationships and attributes. Therefore, we do not consider this information in the transformation model. Two EA models for the same point in time cannot have a transformation model. The same current and target architecture can have more than one transformation model. A successor relationship shows that an element has a successor element in the EA model for the later point in time. An element can be a successor of itself if it is part of the current as well as the target architecture.

A successor relationship is transitive and always connects two elements of the same type. Aier and Gleichauf (2010a) introduced six different types of successor bundles that we explain in Subsection 6.1.2. We extend the types for the cases where one element is part of the bundle in the current and target architecture. This is the case for the oneToMany, manyToOne, and manyToMany types. Figure 3.2 shows the extended bundles. The transformation model limits the number of possible transformation paths. It is not possible to have a transformation model without a target architecture.

With this definition of successor relationships at hand it is now possible to show the different connection between the introduced concepts. Figure 3.3 shows the combinations of current architecture, target architectures and transformation models. Furthermore, it shows some impossible relationships with respect to our definitions given.

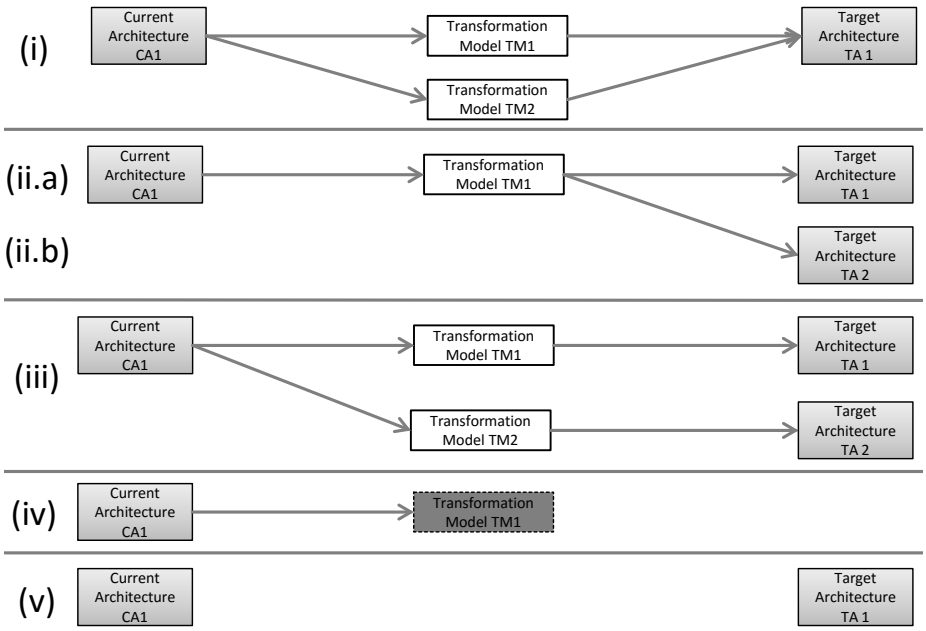

Figure 3.3: Possible and Impossible Relationships Between Current Architecture, Transformation Model(s), and Target Architecture

A current and target architecture may be connected through different transformation models (Figure 3.3, (i)). Regarding the elements of a target architecture it is not possible that the same transformation model connects different target architectures (Figure 3.3, (ii.a)). It is still possible to have target architectures that differ in relationships and attributes present in them, but are connected via the same transformation model (Figure 3.3, (ii.b)). Nevertheless, they have the same direction according to our definition. Different transformation models may also connect the same current architecture with different target architectures (Figure 3.3, (iii)). It is not possible that a current architecture has a transformation model, but no target architecture is available ((Figure 3.3, (iv))). However, it is possible that a current and target architecture have been modeled, but no transformation model is available (Figure 3.3, (v)). In this case the transformation model needs to be reconstructed to allow for a transformation path generation (Lautenbacher et al., 2013; Aier and Gleichauf, 2010a). The transformation paths for different transformation models can never be the same, because different transformation models imply different successor relationships that result in different necessary changes.

Furthermore, we define a transformation path as a semi-ordered sequence of transformation actions that contains at least all necessary changes specified in a target architecture from a given current architecture. Two transformation paths consisting of the same set of transformation actions but with different sequences for the transformation actions are considered as different transformation paths. The number of all possible transformation paths is limited through the target architecture and modeled transformation actions.

3.2 Formalization of the Concepts and Introduction of Transformation Patterns

So far we have neither introduced any EA metamodel nor formalization because all the presented definitions are metamodel independent. However, a formalization is necessary to realize an interactive transformation path generation, as automated planners are only able to create plans that make sense, given a formal description of the domain, the state of the world and actions that allow the planner to change the states.

According to Binz et al. (2012b) an enterprise topology, as an instance model of the enterprise, can be formalized using graphs and abstracted to EA models. Several other approaches exist that also use graphs for formalizing (parts of) an EA model ((Ernst, 2010, p. 53), (Op 't Land, 2008), (Postina, 2012)). Given the expressiveness of graph formalisms we argue that typed attributed graphs can be used to formalize an EA model.

We define the graph of an EA model for a point in time as:

Definition 12. *$EAG_{pointInTime} = (AG, t)$ as a typed attributed graph with:*

- *an attributed graph morphism $t : AG \to ATG$, where ATG is an attributed type graph*
- *an attributed graph $AG = (G, D)$ with:*
  - *$G = (V_G, V_D, E_G, E_{NA}, E_{EA}, (source_j, target_j)_{j \in |G,NA,EA})$ as an E-Graph with:*
    - *V_G and V_D as the finite sets of graph nodes and data nodes present for $pointInTime$*
    - *E_G, E_{NA},E_{EA} are finite sets of edges with source and target functions that hold for $pointInTime$*
    - *$source_G : E_G \to V_G$, and $target_G : E_G \to V_G$ for the graph edges that hold for $pointInTime$*
    - *$source_{NA} : E_{NA} \to V_G$, and $target_{NA} : E_{NA} \to V_D$ for node attribute edges that hold for $pointInTime$*
    - *$source_{EA} : E_{EA} \to E_G$, and $target_{EA} : E_{EA} \to V_D$ for edge attribute edges that hold for $pointInTime$*
  - *D as a $DSIG$-algebra*

Even though a target architecture may not be attached to a certain point in time it is determinable to a point in time. This is in contrast to a vision (architecture) that serves as a guidance for developing a target architecture but is not attachable to a point in time, as it is not defined in terms of a state (c.f. Aier et al. (2009, p. 65) and Buckl and Schweda (2011, p. 18)). Nevertheless, such information can also be modeled in an EAG.

We introduce $EAG_{current}$ as the current enterprise architecture and EAG_{target} as the target enterprise architecture. We limit the elements of $EAG_{current}$ and EAG_{target} to those present at that certain point in time. For example we do neither consider already retired applications and their dependencies as elements of $EAG_{current}$ nor consider planned applications as elements of $EAG_{current}$. The latter are part of EAG_{target} as well as the elements that are not planned to be replaced.

Every EAG is typed by an attributed type graph. As a consequence an EA metamodel is formalized using a attributed type graph that we will refer to as $EATG$.

Definition 13. *We define the attributed type graph $EATG = (TG, Z)$ as the attributed type graph of the EA metamodel, with*

- *a type graph TG*
- *a final $DSIG$-algebra Z*

Even though an EA metamodel may change over time, our assumption is that for a transformation path generation the EA metamodel is stable. Formally, this means that $EAG_{current}$ and EAG_{target} are typed over the same $EATG$.

Furthermore, we abstract EAG to a set. This is possible as a graph consists of a set of edges and nodes (Jungnickel, 2013). The set operators $\cup$ (Union), $\cap$ (Intersection), and $\setminus$ (Complement) are defined as usually. A union of two sets creates a new set containing all elements of both sets. In contrast the intersection of two sets creates a new set that contains only the elements present in both sets. Using the complement set operator for two sets creates a set that contains only elements present in the complemented set. If we complement a set A with B by stating $A \setminus B$, A is the complemented set and contains only the model entities present in A but not in B. We then define

Definition 14. *$EAG_{pointInTime}$ is a set $M := V_G \cup V_D \cup E_G \cup E_{NA} \cup E_{EA}$, containing the nodes and edges of every $EAG_{pointInTime}$.*

Given the sets for $EAG_{current}$ and EAG_{target} we define:

Definition 15. $onlyCurrent := \{x \mid \forall x : x \in EAG_{current} \wedge x \notin EAG_{target}\}$

Definition 16. $onlyTarget := \{x \mid \forall x : x \notin EAG_{current} \wedge x \in EAG_{target}\}$

Definition 17. $stable := \{x \mid \forall x : x \in EAG_{current} \wedge x \in EAG_{target}\}$

This allows us to state that $(onlyCurrent \cap stable \cap onlyTarget) = \emptyset$. Furthermore, the following condition holds: $\forall x : x \in stable \Rightarrow x \in EAG_{current} \wedge x \in EAG_{target}$. In each set x is either a node or an edge. Given a current and target architecture each edge and node belongs to exactly one of the above sets. $onlyCurrent$ contains the gaps that only exist in the current architecture. In contrast, $onlyTarget$ contains the gaps that only exist in the target architecture. We define $stable$ as the set of elements, relationships, and attributes that exist in the current and target architecture. They can be considered as 'negative-gaps'. Given these definitions we are able to determine which parts of the models change and which remain stable. One may argue that there is only one EA model for the present point in time containing all information from the current and target enterprise architecture. This can be correct depending on the definition of EA model. Our view using different EA models for different points in time however, is necessary to allow for a distinction between elements, relationships and attributes that may change and can be reconstructed from the one-model viewpoint by limiting the entities considered within $EAG_{current}$ and EAG_{target}.

So far, we are not able to state how the gaps relate to each other. Therefore, we introduce successor relationships for the element nodes. We therefore define a transformation model connecting a current and target architecture and containing all successor relationships as:

Definition 18. *$transformationModel = (V_G, E_{succ}, source, target)$ is a graph with the following characteristics:*

- *$V_G = (V_{G_{current}} \cup V_{G_{target}})$ is the set of graph nodes from the current and target architecture, such that $V_{G_{current}} \subseteq EAG_{current}$ and $V_{G_{target}} \subseteq EAG_{target}$*
- *E_{succ} is the set of successor edges disjoint from $(E_i \cup E_j)$ such that $E_i \subseteq EAG_{current}$, $E_j \subseteq EAG_{target}$ and $i, j = G, NA, EA$ for all different edge types*
- *$source : E_{succ} \rightarrow V_G$ with $((V_G \cap EAG_{current}) = V_G) \wedge ((V_G \cap EAG_{target}) = \emptyset)$*
- *$target : E_{succ} \rightarrow V_G$ with $((V_G \cap EAG_{current}) = \emptyset) \wedge ((V_G \cap EAG_{target}) = V_G)$*

Formalized actions allow an automated planner to compute possible transformation paths. We call them transformation actions. A transformation action is formalized as a typed attributed graph production with a LHS, RHS, and NAC (see Section 2.4) and specifies a change that can be considered in transformation path planning. A transformation path consists of a sequence of typed attributed graph transformations, i.e.

applicable transformation actions. Transformation actions can be alternatives to each other and can be a composite of other transformation actions. They are introduced in the chapters 4 and 5.

To be able to structure the modeled situations and necessary transformation actions we introduce transformation patterns. A transformation pattern is the abstraction of modeled situations that only differ in the individuals involved in the model. It is formalizable via a typed attributed graph. As a consequence all modeled situations that are abstracted to the same transformation pattern are given through a typed attributed graph morphism. Figure 3.4 shows the transformation pattern created through an abstraction from two modeled situations, where two applications are a successor of themselves.

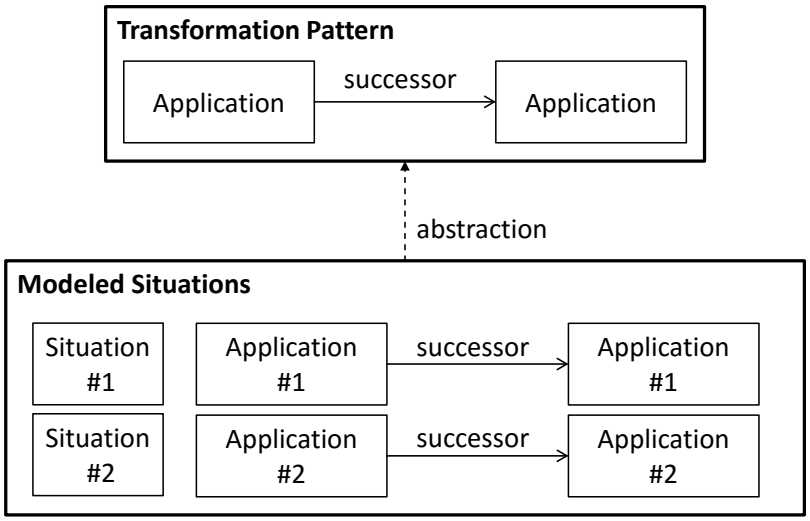

Figure 3.4: Two Modeled Situations and Their Abstracted Transformation Pattern

By choosing an EA metamodel and the elements, relationships, and attributes to be planned the available transformation patterns are determined. The same transformation pattern may allow the execution of transformation action alternatives. But also the same transformation action may be applicable to different transformation patterns.

3.3 Restriction of Transformation Patterns

The number of existing transformation patterns is per se limited by the $EATG$ used for modeling, successor relationships bundles, $onlyCurrent$, $onlyTarget$, and $stable$. We restrict the number of possible transformation patterns again by focusing on aspects relevant from practical points of view and take them into account within the planning domain model.

3.3.1 Feedback Loops Viewpoint

We start restricting the possible transformation patterns by introducing a feedback loop viewpoint on transformation planning. The concept of feedback loops has its origins within the systems theory of closed control loop systems. An actuator computes (plan) the changes to the system's parameters and executes (do) them in terms of controllable variables. A controllable variable of the system engine is for example rounds per minute. After some time has passed it is possible to check whether the desired output, e.g. an increase in rounds per minute, was accomplished successfully and if the success was exactly as desired. Given the feedback information it is now possible for the controller to determine (check) whether the computed input resulted in the desired output, and if not, to adjust (act) the variables for steering the system. The enterprise can be considered as a system which needs to create feedback information in order to allow for an improvement of the system's performance within the controllable boundaries.

We create the feedback loop viewpoint by introducing a new architecture state that is the new current architecture at a future point in time: $EAG_{current_{t1}}$, where $t1$ becomes the present point in time and $t0$ is in the past. The motivation of this viewpoint is that in EA planning at a certain point in time the enterprise architect checks whether the changes were carried out as planned or if not which were not. This allows to create a feedback loop investigating what the reasons were for deviations from the intended transformation. The viewpoint is time dependent which means that for a point in time (t) when the viewpoint is created the $EAG_{current_{t1}}$ is the current architecture (for t) and the former current architecture ($EAG_{current_{t0}}$ for t-1) becomes the $EAG_{current_{t1}}$ and the former target architecture ($EAG_{target_{t0}}$ for t-1) becomes the $EAG_{target_{t0}}$. The underlying assumption is that $EAG_{target_{t0}}$ was supposed to be changed into $EAG_{current_{t1}}$. If this is not the case, $EAG_{target_{t0}}$ should be restricted accordingly. By using the set operators we are now able to create seven subsets, which we describe in the following. Figure 3.5 shows an overview of the different sets and subsets of the feedback loop viewpoint. Please note that we named the subsets alphabetically in order to keep Figure 3.5 compact. An alternative naming would have been to call for example subset A: *was planned to be removed and was removed.*

With $EAG_{current_{t0}}$, $EAG_{target_{t0}}$ and $EAG_{current_{t1}}$ at hand we define the seven subsets A to G.

Subset A contains elements, relationships and attributes which belong to the $EAG_{current_{t0}}$ but are neither in the $EAG_{target_{t0}}$ nor $EAG_{current_{t1}}$. An example could be an application which was retired as planned until the $EAG_{current_{t1}}$ was created.

Definition 19. *Subset A is defined as*
$A = EAG_{current_{t0}} \setminus (EAG_{target_{t0}} \cup EAG_{current_{t1}})$

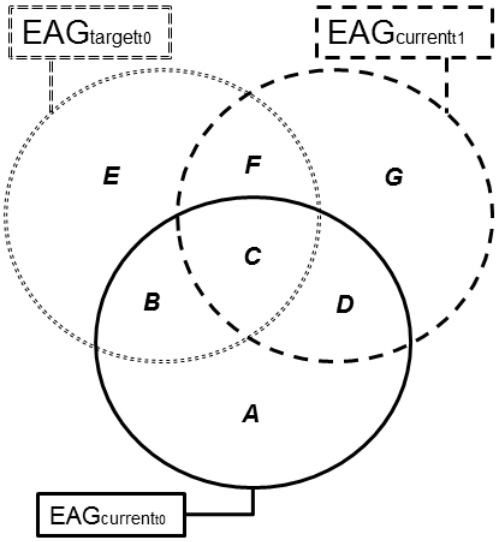

Figure 3.5: Different Subsets Derivable from the Sets $EAG_{current_{t0}}$, $EAG_{target_{t0}}$ and $EAG_{current_{t1}}$

Subset B contains elements, relationships and attributes which the $EAG_{current_{t0}}$ and the $EAG_{target_{t0}}$ have in common, but do not belong to the $EAG_{current_{t1}}$. These were planned to be present in the future, however they are not. An example could be an application that was not planned to be retired, however it is was retired.

Definition 20. *Subset B is defined as*
$B = (EAG_{current_{t0}} \cap EAG_{target_{t0}}) \setminus EAG_{current_{t1}}$

Subset C contains elements, relationships and attributes which were not planned to change and remained stable. An example could be an application which was present in $EAG_{current_{t1}}$, was planned not to be phased out and is still in use.

Definition 21. *Subset C is defined as*
$C = EAG_{current_{t0}} \cap EAG_{target_{t0}} \cap EAG_{current_{t1}}$

Subset D contains elements, relationships and attributes which were planned to be changed, however they remained unchanged as they belong to $EAG_{current_{t0}}$ and $EAG_{current_{t1}}$, but not to $EAG_{target_{t0}}$. An example could be an application that was planned to be retired, however it is still in use.

Definition 22. *Subset D is defined as*
$D = (EAG_{current_{t0}} \cap EAG_{current_{t1}}) \setminus EAG_{target_{t0}}$

Subset E contains elements, relationships and attributes which were planned to be changed, but were not. An example could be an application that was planned to be developed, however it is not yet ready for use.

Definition 23. *Subset E is defined as*
$E = EAG_{target_{t0}} \setminus (EAG_{current_{t0}} \cup EAG_{current_{t1}})$

Subset F contains elements, relationships and attributes which were introduced as planned. An example could be an application that was planned to be developed and is in use as planned.

Definition 24. *Subset F is defined as*
$F = (EAG_{target_{t0}} \cap EAG_{current_{t1}}) \setminus EAG_{current_{t0}}$

Subset G contains elements, relationships and attributes which were developed, but this was not considered in the plan. An example could be an application that was not planned to be developed, but is now in use.

Definition 25. *Subset G is defined as*
$G = EAG_{current_{t1}} \setminus (EAG_{current_{t0}} \cup EAG_{target_{t0}})$

Given these subsets we argue from a transformation path planning perspective that not every type of gap is planned, even though the EA models would provide means to do so. These potential gaps are elements, relationships and attributes in subset C that never change. Together with the elements, relationships and attributes that are planned to remain stable they form the subset C.

Furthermore, we state that from a planning perspective types of gaps exist which could be created, but are not considered as changes which have to be planned in transformation path generation. An example would be a strategic goal which may change due to changes in the strategy of the enterprise. However, such a change is not within the scope of transformation paths. Such a strategic change is the trigger for a transformation of the enterprise rather than something that is planned as part of transformation paths.

3.3.2 Scoping the Transformation Paths

In the following we determine the scope of the transformation paths. We will focus on the changes of applications and their application services. However, those two elements and their interdependencies would not be enough to generate transformation paths. They are the elements which are to be planned but we need also means for selecting a target architecture and ranking different transformation actions that result in it. We will introduce the necessary concepts in the following and consider them in the requirements for the planning domain model.

Business Support Maps as a Means for Target Architecture Selection Business support maps are a common means for aligning applications to the business. Figure 3.6 shows an example for a business support map with business processes, organizational units, and applications. From left to right are two business processes for customer management and order fulfillment shown. Figure 3.6 depicts on the left from top-down three organizational units: headquarters and two subsidiaries. Furthermore, the business support map shows which application is used in which organizational unit in which business process.

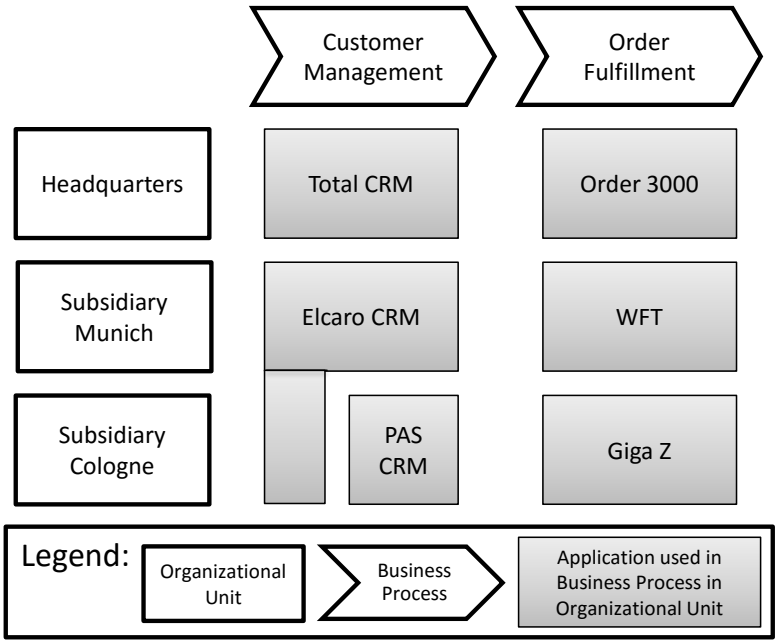

Figure 3.6: Example of a Business Support Map

Business support maps are used for planning purposes and several business support maps are created for different points in time. We call a business support map capturing the status quo: current business support map. It serves as the starting point for planning purposes. In contrast target business support maps are used to model the desired future support. The current and target business support maps are compared to identify necessary changes (Hanschke, 2013). However, this is only true for the level of detail specified within the business support maps and where an explicit replacement is modeled.

Enterprise architects use current business support maps for creating transparency by gathering information on application services and applications currently used in business processes. By doing so an enterprise architect is able to identify redundancies of application services and applications used for the same process activity or process activities without IT support. Figure 3.6 shows a redundancy in the business support of the application Elcaro CRM for the subsidiary in Cologne in the business process customer management.

Birkmeier et al. (2013) describe how to define appropriate services derived from process models in the context of an EA. It is possible to focus on specific business processes to plan the IT support in detail, i.e. to decide which process activities are to be supported by which application services. The underlying assumption is that the IT support for the other business processes is not planned in detail.

Within target business support maps enterprise architects express the desired future which should be realized. Typical changes compared to the corresponding current business support map are the removal of redundancies, the population of empty cells in the business support maps to plan future IT implementation or explicit replacement. These changes correspond to typical IT transformation changes: create, modify and delete (Simon, 2009).

Translating Strategic Directives into Concrete Changes Strategic directives serve as a frame for deriving decisions that result in concrete changes (Hanschke, 2013, p. 80). However, it is important that strategic directives do neither restrict the decisions to much nor leave to much freedom of choice in steering changes. Logical functionalities are a common means to steer planning efforts, by grouping applications and their functionalities into an implementation and technology independent way. An example for a logical application is a customer relationship management tool. An application used in the enterprise would be a concrete product from a vendor which provides an implementation of a customer relationship management tool. Logical functionalities are abstractions of implementations that are more fine grained than logical applications. An application service implements a functionality that can be abstracted to an implementation independent level which allows to compare those logical functionalities. By using these implementation independent concepts an enterprise architect is able to define some structure for a target architecture without making any constraints on applications and application services which are to be used for implementing them. How to derive strategic directives in the context of EAM is discussed in more detail in Op 't Land et al. (2009, p. 31) and Hanschke (2013, p. 330).

Tactical Planning Concerns From a practitioner's point of view we need to further narrow the planning entities. EAM does not plan business processes or the deployment of applications. It gathers information on changes that are planned in business processes and how the supporting IT is influenced by these changes. Based on this information EAM supports planning the conjoint development of business and IT for different time horizons. The time horizons are differentiated depending whether the planning is concerned with strategic, tactical or operational planning. Strategic planning, in the context of EAM, is concerned with creating a shared understanding, a so called vision, of where the business and IT want to be in the long term future Hanschke (2013, p. 75). It has the longest time horizon in contrast to operational planning.

Operational planning is concerned with detailed planning issues like project milestones. Between strategic and operational planning lies the tactical planning which covers the concretization of the strategic planning results, by deriving necessary changes and their temporal dependencies. By focusing on tactical planning concerns we limit the scope of the transformation paths and as a consequence the planning entities. A typical time frame for tactical planning is three to five years in the future (Hanschke, 2013, p. 614).

3.4 Requirements for the Planning Domain Model

In the following we introduce the requirements which serve as the basis for the design of the planning domain model. The first set of requirements are derived from the state of the art in literature. The second set are requirements that derive from the restrictions on the transformation patterns and the scope we set for the transformation paths. The third set of requirements only contains one requirement that is necessary to allow for an automated planner to support in the generation of the transformation path.

I: Requirements derived from state of the art in literature

IR1: The planning domain model MUST provide elements, relationships, and attributes for current and target architectures on the same level of detail (derived from Hanschke (2013, p. 73 ff.) and Buckl et al. (2008, p. 64))

IR2: The planning domain model MUST provide elements, relationships, and attributes for sequencing transformation actions (derived from Aier and Gleichauf (2010b, p. 6), Clark et al. (2011, p. 94), and Agievich and Skripkin (2014, p. 234))

IR3: The planning domain model MUST consider elements, relationships, and attributes that allow for a distinction between possible efforts for transformation actions (derived from Teece (2007, p. 1327) and Zimmermann (2008, p. 462))

II: Requirements from restrictions of transformation patterns and scoping of transformation paths

IIR1: The planning domain model MUST provide an explicit transformation model for the applications and application services

IIR2: The planning domain model MUST allow for a frame setting mechanism to allow for a reduction of the number of possible target architectures by considering strategic directives within business support maps

IIR3: The planning domain model MUST allow the localization of application and application services within the business support maps to enable top-down and bottom-up planning

IIR4: The planning domain model MUST allow for an explicit replacement of applications and application services

III: Requirement demanded from automated planning formalisms

IIIR1: The planning domain model MUST provide extra data for the automated planner to mark goal states in the graph transition systems used for transformation path planning

3.5 Planning Domain Model for Interactive Transformation Path Generation

In the following we introduce the planning domain model for interactive transformation path generation and its formalization via a type graph (see Figure 3.7 on page 58). The design process for the planning domain model followed the guidelines from Vaquero et al. (2011) as described in Subsection 2.7.1.

3.5.1 Elements, Relationships, and Attributes

We consider current and target business support maps at a coarse level of detail for business processes and organizational units and at a detailed level for process activities and roles of employees. A business process is a semi-ordered set of process activities and has a clear defined start and end in terms of input and output. Process activities are the partial tasks of a business process that transform the given input into the specified output. They are carried out by employees that have a certain role in a certain activity. An organizational unit is a way to structure the enterprise into hierarchies of competencies and duties. It has its own resources and goals.

An application is a software that is used by employees in a business context and supports them in their tasks by digitalizing parts thereof or even automating parts thereof. We will not distinguish between commercial of the shelf, customized commercial of the shelf or individual developed applications within the metamodel. A commercial of the shelf application is one that can be bought or rented from a vendor and used without any further development efforts. When a commercial of the shelf application is extended with additional functionality or adaptations of existing functionality takes place the

application is called customized. Individual developed applications are software that mainly consists of code that is not publicly available. Within the last decade the advent of service oriented architectures as a paradigm for loose coupled applications has been introduced. It allows one to orchestrate, compose and reuse so called application services without causing direct dependencies between applications. This allows to change an application, that implements an application service, without a consequence for the applications that use the application service.

Furthermore, applications and application services have lifecylce phases that capture the state of them (**IR1** and **IR2**). We differentiate between the following lifecycle phases: live, proposed, and retired. Planned elements are going to be realized. Being in a lifecycle phase live means that the element is ready to use and its development has been finished. After the element was in use its lifecylce phase changes to retired which means that it is no longer in use.

Successor relationships are considered for applications and application services (**IIR1** and **IIR2**). They allow to create the transformation model for the current and target architecture. The successor relationships for other elements are not considered. They are not planned and therefore their successor relationships are implicit. Furthermore, the successor relationships allow for a restriction of possible change sequences, as they indicate which elements need to be developed before others can be retired (**IR2**).

Given the business processes, organizational units, process activities and roles we have defined the elements that allow us to localize others. As business support maps need to be transformed from a graphical representation, with a x-axis and a y-axis, into a model that represents the ternary relationship a special kind of element is suggested by Buckl et al. (2009b). We call these elements business support elements, respectively current and target business support elements. The actual elements in the business support maps are then localized via the business support elements' corresponding business process, process activity, organizational unit and role. Therefore, they allow to support the selection of a target architecture (**IR1**). Each current business support element has a corresponding target business support element. Applications and application services are localized by current and target business support elements (**IIR3**). They are localized by target business elements to allow for direct replacement or to state that the business support remains stable (**IIR4**).

A logical application is an abstraction of an application from its implementation. It is modeled in target business support maps and does not explicitly specify an application to be used in the target architecture (**IIR2**). However, their usage limits the design decisions necessary for target architectures to all applications that are an implementation of the logical application or a new one.

The counterpart of the logical applications regarding services are the information services. We use information services as an implementation independent concept of an

application service to express its functionality. Information services are therefore the frame setting mechanism within the target business support maps (**IIR2**). The functionality of an application service can be enhanced and decreased.

Definition 26. *We define the functional enhancement of an application service as realizing a new information service.*

Definition 27. *We define the functional decrement of an application service as the removal of a realization of an information service.*

An application service that does not realize any information service cannot be functional decremented. Vice versa an application service that already realizes every information service cannot be enhanced. Furthermore, we consider information objects that are accessed through the information services. These objects allow for communication between business and IT. Information objects are detailed enough to specify their content, but are implementation independent. The content is determined in terms of schema entries that reflect the complexity of the information object (**IR3**). Information services can access information objects in different ways. An information service can create, read, update or delete an information object. We consider an information object and information service as virtual artifacts. Virtual artifacts are neither part of the business nor the IT architecture and are used for decoupling both architectures (Aier and Winter, 2009). Information services and logical applications are a means for an enterprise architect to model a demand for IT support within a process activity raised by a stakeholder.

3.5.2 Planning Domain Model Fragments for the Automated Planner

A goal element type and reached attribute type are model fragments of the planning domain model for an automated planner to mark states that are goal states (**IIIR1**). Between current and target business support elements a 'target' edge type allows to point from a current to the corresponding target business support element. This allows to model transformation actions that result in less computational efforts for the automated planner, as only the target edge is relevant and not the whole context of each business support element. Introducing the edge types 'to be enhanced' and 'to be decremented' is necessary to take functional enhancements and decrements of application services into account (see Definitions 26 and 27). An attribute type marker allows the automated planner to tag applications and application services during planning to prevent an unwanted retirement of elements. Lifecyclephases of the applications and application services are also important for the automated planner as they allow to apply changes to them during transformation planning (**IIIR1**). However, the lifecycle phases are also relevant for the enterprise architect in contrast to the attribute type marker. Please note that all of the data necessary for the automated planner is generated automatically and does not require manual input.

3.5.3 Formalization of the Planning Domain Model

The planning domain model is formalized via the attributed type graph PDM as shown in Figure 3.7. PDM contains node types, edge types, and node attributes that are necessary for capturing the domain knowledge and allow for interactive transformation path generation.

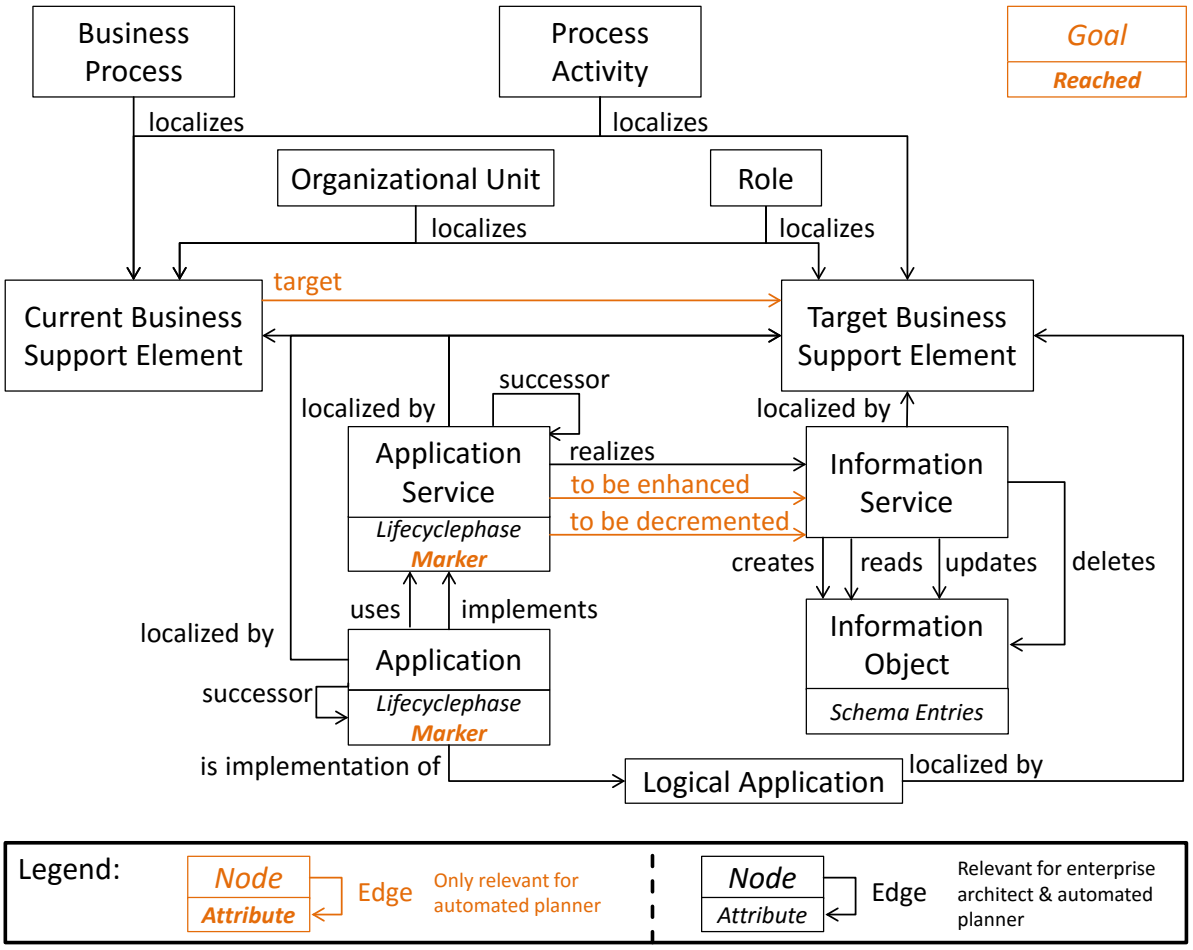

Figure 3.7: Type Graph PDM for the Planning Domain Model

Each box in Figure 3.7 is a node type and each connection between them is an edge type. Node attributes are at the bottom of the boxes and have an italic font. Nodes, edges, and attributes color-coded in orange are only relevant for the automated planner. All other nodes, edges, and attributes are relevant for the enterprise architect, as well as the automated planner. Regarding the parts of it that are relevant for the enterprise architect it can be considered as a subgraph of an $EATG$. This may be not true for every $EATG$, as for example an information service is a rather sophisticated and detailed concept in EA models. An $EATG$ may be extended to satisfy the condition of $PDM \subseteq EATG$.

3.5.4 Final Data Signature $DSIG_{PDM}$

In the following we informally introduce the final data signature $DSIG_{PDM}$, that allows us to create the graph transformation systems used in the subsequent Chapters 4 and 5. Figure 3.8 shows the $DSIG_{PDM}$ part that determines the values of the attributes as shown in Figure 3.7.

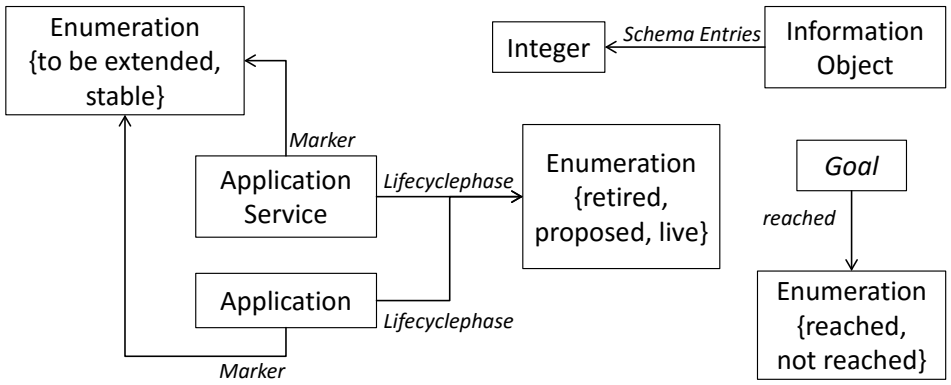

Figure 3.8: Final Data Signature $DSIG_{PDM}$ for the Planning Domain Model

We use enumerations to restrict the possible values for certain attribute types (c. f. Object Management Group (2009, p. 100)). The marker values are limited to 'to be extended' and 'stable'. This allows the automated planner to mark applications and application services that belong to *stable* and if the application, respectively application service, is going to provide more functionality.

The lifecycle phase values are restricted to the phases 'retired', 'proposed', and 'live'. If the lifecycle phases for the applications and application services do not reflect the lifecycle model for those two elements the enumerations can be adapted. The design of the transformation actions later must consider that the lifecycle phases are not violated, regarding the semantics of the lifecycle phases. The semantics of the lifecycle phases could be expressed, for example, with a state machine.

The last enumeration present in $DSIG_{PDM}$ restricts the values of the reached attribute of goal nodes to 'reached' and 'not reached'. Therefore, we can explicitly mark the goal states where a goal is reached and where not. The schema entries of information objects are restricted to integers and allow the calculation of the efforts later on.

3.6 Reusing Knowledge Base Information

After specifying the planning domain model we proceed with the task of modeling actual planning problems. As we want to reuse as much as possible of the existing knowledge we follow the approach of Semantic Enterprise Architecture Management for integrating domain knowledge into a coherent EA model and metamodel using semantic web technologies (Chen et al., 2013).

Based on this model a strategic planning takes place which forms the foundation for setting a frame for the target architecture using business support maps. Using semantic web technologies for modeling has the advantage that consistency checks and reasoning over the knowledge base are facilitated by a reasoner. Thus, it is possible to query a consistent and coherent knowledge base that satisfies the information needs of enterprise architects.

Technically the used terminological box of the used EA metamodel needs to be mapped into the type graph to allow a sound model transformation later on. The shortcoming of this mapping are the efforts necessary to specify the mapping and to execute the model transformation. However, the consistency of the EA models and the materialization of implicit knowledge is ensured a priori within the knowledge base and allows us to assume consistent models regarding the formalisms. The resulting benefit is that the EA models, formalized with semantic web technologies, provide ease of use and at the same time allow for interactive transformation path generation in the background, using typed attributed graph transformations.

Mapping of Planning Domain Model Elements to other EA Metamodels

We provide a mapping to the planning domain model allow different EA metamodels and respective EAM approaches to facilitate the reusability of our approach. Table 3.1 shows the mapping of planning domain elements to elements from the TOGAF Content Metamodel (The Open Group, 2011, Chapter 34), the Best-Practice EA (Hanschke, 2013, p. 204 ff.), and the EAM Pattern Catalog metamodel (Buckl et al., 2008, p. 188 ff.). Please note that the latter metamodel is called information model from its authors.

The planning domain elements ‘application’, ‘application service’, ‘business process’, and ‘organizational unit’ all have a corresponding element in the other metamodels. This is a consequence of the fact that these elements can be considered as essential artifacts of EA models Winter and Fischer (2006) and at the same time are important for transformation path generation. ‘current business support element’, ‘target business support element’, ‘role’, and ‘process activity’ have corresponding concepts in two other

Planning Domain Model Element	TOGAF Content Metamodel (The Open Group, 2011, Chapter 34)	Best-Practice EA (Hanschke, 2013, p. 204 ff.)	Information Model of EAM Pattern Catalog (Buckl et al., 2008, p. 188 ff.)
Application	Physical Application Component	Information System	Business Application
Application Service	Information System Service	Interface	Application Service
Business Process	Process	Business Process	Business Process
Current Business Support Element	x	Functional Assignment	Support Relationship
Information Object	x	Information Object	x
Information Service	x	x	x
Logical Application	Logical Application Component	x	x
Organizational Unit	Organization Unit	Organizational Unit	Organizational Unit
Process Activity	Process	x	Activity
Role	Role	x	Business Role
Target Business Support Element	x	Functional Assignment	Support Relationship

Table 3.1: Mapping of Planning Domain Model Elements to other EA Metamodels

metamodels. TOGAF does not consider business support elements, whereas the Best-Practice EA does not consider roles and process activities.

Please note that the planning domain model elements 'business process' and 'process activity' map both on the TOGAF's process concept. This is possible because TOGAF allows for a (recursive) refinement of processes through processes.

The only metamodel that provides a concept comparable to the 'logical application' is the TOGAF Content Metamodel. It considers a logical application component as an "encapsulation of application functionality that is independent of a particular implementation" (The Open Group, 2011, p. 357). An 'information object' is only explicitly considered the Best-Practice EA (Hanschke, 2013, p. 219). Nevertheless, information objects have similar concepts in the TOGAF Content Metamodel (data entity) and in the EAM Pattern Catalog metamodel (business object), that could be used to derive, respectively aggregate, information objects. The only concept of the planning domain model that does not have a mapping is the 'information service'.

3.7 Summary of Transformation Patterns and Planning Domain Model

In this chapter we introduced the concepts involved in transformation path planning and the relevant model contents to be reused in the subsequent chapters. We proceeded with a formalization of the concepts and introduced transformation patterns to abstract from modeled situations to a common pattern. Additionally, we narrowed the set of transformation patterns to a reasonable and relevant subset. A restriction was established by introducing a feedback loop viewpoint on transformation paths and by focusing on decisions involved in tactical planning concerns. The subset of transformation patterns to be considered, was then internalized to the planning domain model, that is an EA metamodel. We formalized it with a type graph, that serves as the basis for the formalization of transformation actions in the subsequent chapters. Furthermore, we presented a mapping of TOGAF's, Best–Practice EA's, and the EAM Pattern Catalog's metamodels to the planning domain model to allow for a reuse of existing knowledge base information, that may be based on these metamodels. Furthermore, the mapping showed the nearness of the planning domain model to these widely used metamodels and how to cope with deviations.

The design of the artifacts presented in this chapter implies some restrictions on the artifacts that are to be introduced in the subsequent chapters as the design decisions made in this chapter have an impact on these artifacts in the following chapters. Regarding the selection of a target architecture we state that goal states represent different tar-

get architectures. For the transformation path generation a goal state is given through the target architecture and transformation model. Additionally, different sequences of transformation actions need to be supported to determine the actual order of the transformation actions involved in the transformation path.

4 Interactive Decision Support for Target Architecture Selection with Business Support Maps

In the following chapter we present our approach for interactive decision support for target architecture selection with business support maps. Two different approaches are supported. On the one hand we allow for a bottom-up approach using business support maps with application and information services as elements being localized. On the other hand we allow for a top-down approach where applications and logical applications are localized within the business support maps. Even though the approaches have variation points they share some commonalities. We describe necessary input from an enterprise architect for both approaches and the underlying formalisms. Furthermore, we show how transformation actions are computed, ranked and can be applied in different resource modes. The latter allows to consider the desired speed of change for certain transformation actions. We end the chapter with information on how the target architecture is finalized and a summary.

4.1 Overview of Bottom-Up and Top-Down Approach

In the following we discuss the commonalities of the different approaches used for selecting a target architecture. Furthermore, we present the assumptions regarding the EA models' content. Figure 4.1 shows an overview of the commonalities of both approaches.

The input is necessary to start the process of selecting a target architecture. After a target architecture is selected the output of this chapter will be used as the input for Chapter 5 Interactive Transformation Path Generation.

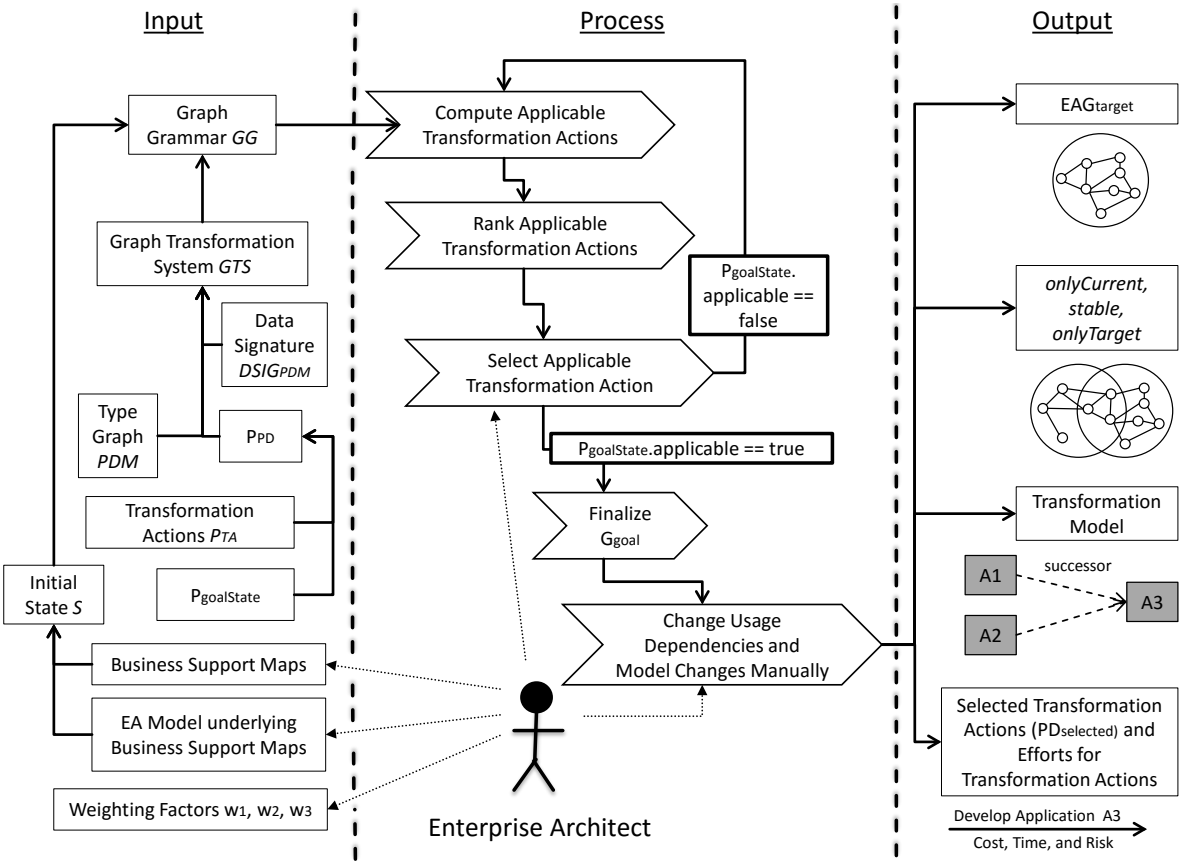

Figure 4.1: Overview of Input, Process, and Output for Interactive Decision Support for Target Architecture Selection Through an Enterprise Architect

4.1.1 Input from Enterprise Architects and Formalisation of Both Approaches

We need a graph transformation system GTS and an initial state S to create a graph grammar GG that allows us to explore possible target architectures. The initial state S is queried from the EA tool in order to reuse the existing models. It contains the business support maps and underlying information that we will clarify in the following.

Our assumption is that an enterprise architect has already modeled a business support map for the current and target business support that we can reuse. Furthermore, we assume that the EA model already contains the implicit information underlying the business support maps. For details we refer to the description of the initial states for the top-down and bottom-up approach (see Subsubsections 4.2.1.2 and 4.3.1.2).

Please note that it is possible to neglect these assumptions and create a starting state anyway. However, some of the transformation actions introduced later may need this

information to determine for example if an application service is a candidate to be reused in the target architecture. To complement such incomplete information is out of scope of the thesis.

Furthermore, we need $DSIG_{PDM}$, PDM, and P_{PD} to create a GTS. The attributed type graph PDM is the one as shown in Figure 3.7 and the same for the bottom-up and top-down approach. This is also true for the data signature $DSIG_{PDM}$. However, the set of typed attributed graph productions P_{PD} varies between the approaches as the set of transformation actions $P_{PD_{TA}}$ and goal states vary.

Goal states within GG are identified by the applicability of a special typed attributed graph production $P_{goalState}$, with $P_{goalState} \in P_{PD}$. The application of $P_{goalState}$ does not change the information relevant for the enterprise architect, as it just adds a goal node with a reached attribute. Furthermore, P_{PD} contains typed attributed graph productions that we use to add or remove information from the graph in the background. We describe them in the respective sections.

4.1.2 Computing Applicable Transformation Actions

To be able to check which transformation actions are applicable, we have to compute all LHS of all transformation actions p in $P_{PD_{TA}}$ that have a typed attributed graph morphism, called match m, in G_j. A typed attributed graph transformation is then defined through $G_j \overset{p,m}{\Longrightarrow}{}^{*} G_i$ (Ehrig et al., 2006, p. 182). We call the typed attributed graph transformations: applicable transformation actions. G_j is either S or a G_i created through the applications of a sequence of transformation actions defined through $S \Rightarrow^{*} G_i$. Please note that the NAC of the transformations actions, that prohibit the application of a transformation action, is considered in finding the match in G_j. Finding such a match is in $\mathcal{NP}_{complete}$ (Estler and Wehrheim, 2011, p. 56).

4.1.3 Ranking of Transformation Actions

The ranking of applicable transformation actions allows us to provide an ordering of them for the enterprise architect. Technically the ranking assigns edge weights to the transformations created through GG.

In order to be able to compute the ranking of the transformation actions it is necessary to acquire the preferences from the enterprise architect regarding the different factors used for ranking. Preferences can be derived by either applying ratio comparisons or difference comparisons (Triantaphyllou, 2000, Chapter 4 and 5). To create the ranking of transformation actions we use the Weighted Product Model introduced in Subsection

2.6.3. We calculate the effort for the performance value $R(P_{PD_{TA}})$ of each applicable transformation action $P_{PD_{TA}}$. The effort criteria for transformation actions are cost, time, and risk.

Negative efforts cannot be created, as every change, represented through a transformation action, requires time and money to be implemented. The lower the performance value the better. Each criterion is weighted through a weighting factor and the calculation is done through the following formula:

- $R(P_{PD_{TA}}) = (a_{TA1})^{w_1} \times (a_{TA2})^{w_2} \times (a_{TA3})^{w_3}$ with
- a_{TA1} as the value for dimension cost and w_1 as the weighting factor for the cost dimension
- a_{TA2} as the value for dimension time and w_2 as the weighting factor for the time dimension
- a_{TA3} as the value for dimension risk and w_3 as the weighting factor for the risk dimension

For the cost dimension we consider project costs *and* operation and maintenance costs. These represent the second hierarchical level of the IT cost taxonomy from Närman et al. (2009), as estimations on a very detailed level are difficult to make at this point of decision making. Project costs have fixed costs C_f and may have variable costs C_v. The operation and maintenance costs are either C_s as expected costs for a new application service and C_a as expected costs for a new application. Those costs may be taken from a service catalogue of the enterprise's IT department.

Variable costs depend on factors present in S. The value for the risk dimension is Z_P and assigns to each transformation action a risk value.

4.1.4 Resource Modes for Transformation Actions

Additionally, we allow for the execution of transformation actions in different resource modes. A resource mode determines the number of resources allocated to a transformation action for a faster realization of a change, but the same effect as the transformation action with less resource allocation. We consider the following resource modes: normal, double, and quadruple. We use scaling factors to allow for a compression of the duration of transformation actions and a raise in the costs of the transformation actions. This corresponds to a formalization of the crashing method for compression of project schedules (OConchuir (2011, p. 89) and Project Management Institute (2008, p. 156)).

Crashing methods in project management are used to reduce the duration of projects by the means of adding more resources to the project (Kerzner, 2013, Section 12.5).

The scaling vectors for the values a_{TA1} and a_{TA2} of transformation actions are influenced through the different resource modes. We introduce the following scaling vectors:

- c_d for scaling up the variable costs in the double resource mode
- c_q for scaling up the variable costs in the quadruple resource mode
- t_d for scaling down the variable time in the double resource mode
- t_q for scaling down the variable time in the quadruple resource mode

We assume that doubling the resources does not cut the duration time in half and does not result in the same costs as in a lower resource mode. If the scaling factors would be set negative they could be used to scale down the costs or scale up the time. However, this would not reflect reality in projects.

4.1.5 Selecting Applicable Transformation Actions

After calculating the ranking of each applicable transformation action it is possible to assign edge weights to the transitions between different states generated through GG. Every selected transformation action and its effort values are added to the list $PD_{selected}$, that contains all selected transformation actions. The list is empty at the beginning of the selection process.

After a transformation is selected it needs to be checked if G_i, created through the application, is already present in the set of graphs created through a sequence of applied transformation actions $S \overset{p,m}{\Longrightarrow}^* G_j$. The complexity of checking this isomorphism is in $\mathcal{NP}$ (Estler and Wehrheim, 2011, p. 56).

We allow that an enterprise architect deselects already chosen transformation actions. For example if she has chosen several transformation actions she may decide to go back to S and select different transformation actions that lead her to another goal state.

We differentiate the varying concepts between bottom-up and top-down approaches by adding the abbreviated suffixes BU (for Bottom-Up), respectively TD (for Top-Down) to the terms were necessary.

4.2 Bottom-Up Approach

We introduce the bottom-up approach starting with necessary input. Then, we present the process for the bottom-up approach, followed by the output of the process.

4.2.1 Input for Bottom-Up Approach

In the following we describe the input that we need from an enterprise architect to be able to start the interactive decision support for target architecture selection using a bottom-up approach.

4.2.1.1 Input from Enterprise Architect

Besides, modeled business support maps on a detailed level, it is necessary to retrieve the weighting factors, scaling factors for the resource modes, and the risk value from the enterprise architect to allow for a ranking of the applicable transformation actions.

Bottom-Up Business Support Map Figure 4.2 shows an example for a current and a target business support map in the bottom-up approach. The business process customer project handling consists of several process activities and has two roles involved. For detailed information on the context of the current and target business support map we refer to Subsection 7.2. Entries in the cells of the business support map are:

- empty; no business support modeled
- an application service modeled; and its application in braces implicit
- an information service is modeled in the target business support map; indicated by the star
- a combination of the above
- or an entry is not possible, as the role is not involved in the activity

Modeling the current and target business support maps takes place in the EA tool used by the enterprise architect. The modeled current and target business support maps, including the mentioned implicit information in Subsection 4.1.1, are then exported and transformed into the initial state S_{BU} that is introduced in the following formalisms.

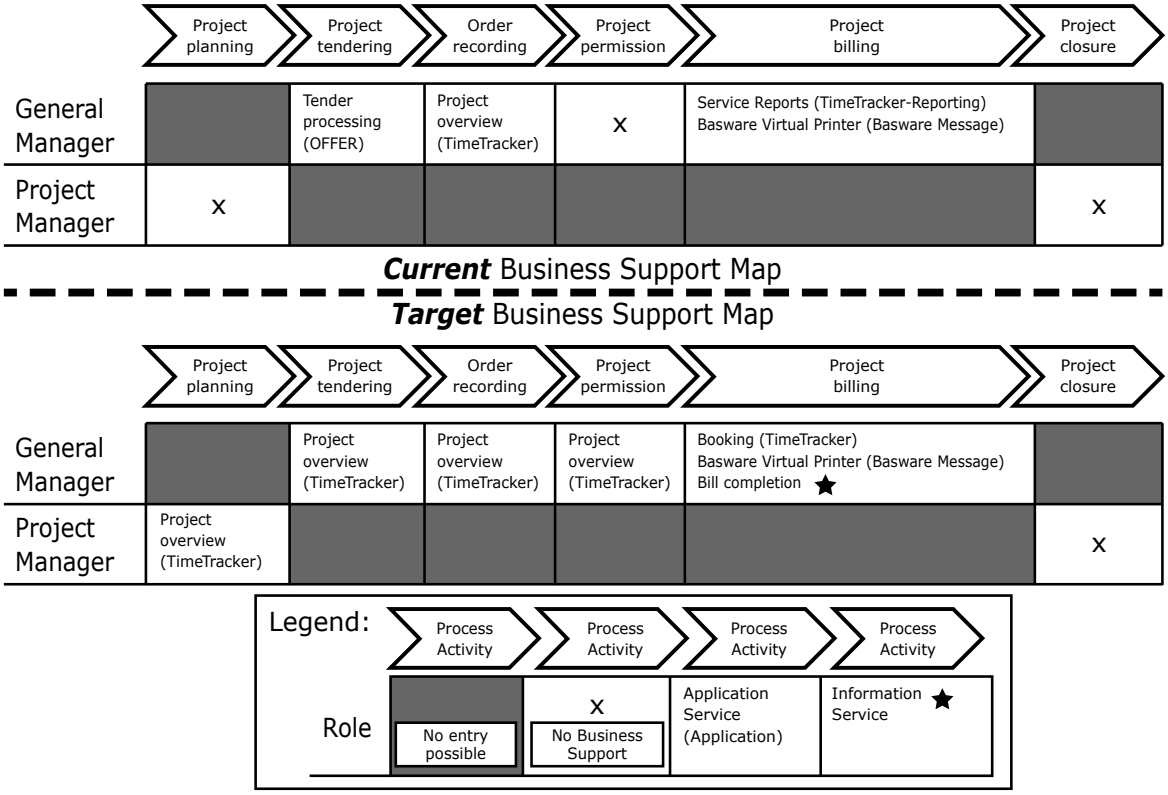

Figure 4.2: Example of Current and Target Business Support Map on Process Activity Level for Bottom-Up Approach

4.2.1.2 Bottom-Up Formalisms

In the following we introduce the formalisms necessary to allow for the interactive selection process. We start with the concerned transformation patterns and transformation actions. Furthermore, we introduce the initial state and the nested typed attributed graph production to detect goal states.

Transformation Patterns Bottom-Up Figure 4.3 shows the transformation patterns we assign a meaning to in the bottom-up approach. Transformation pattern (a) is an abstraction of a situation where an application service is already implemented but currently not in use. A direct replacement of an application service through another already implemented one is expressed through transformation pattern (b). No changes in the business support through an application service is expressed through transformation pattern (c). With transformation pattern (d) a currently not supported activity should be supported in the future. However, modeling this pattern (implicitly) does not state anything about concrete changes to be developed.

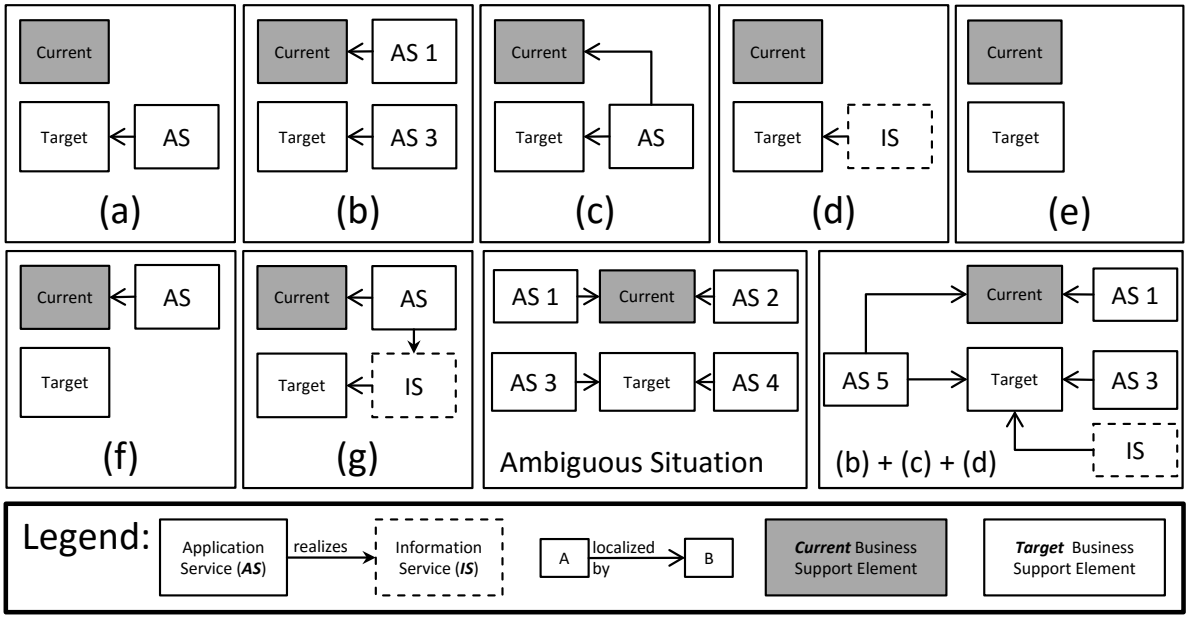

Figure 4.3: Transformation Patterns in Bottom-Up Approach

Transformation pattern (e) is the empty pattern. It states that there is currently no business support and that there should be none in the future. Transformation pattern (f) states that there is a business support through an application service, but there should not be one in the target architecture. A transformation pattern similar to (d) is transformation pattern (g). However, in contrast to (d) it states that there is already a business support through an application service and it should be replaced through another application service. Furthermore, Figure 4.3 shows what we call an ambiguous situation and a combined pattern. The combined pattern is a combination of the transformation patterns (b), (c), and (d). This combination is not ambiguous as the business support of AS 5 is not changed and neither AS 1 nor AS 3 is realizing IS.

In Figure 4.2 the following transformation patterns are present: (a), (b), (c), (d), and (e). For the project billing business process a combined transformation pattern, consisting of (b), (c), and (d) exists. Transformation pattern (a) is present twice in Figure 4.2 for the business process project planning and project permission. In the business process project tendering the transformation pattern (b) exists. Transformation pattern (c) is present in the business process order recording.

Transformation Actions Bottom-Up Transformation actions allow us to derive possible changes to a graph, that an enterprise architect selects. The LHS, NAC, and RHS of all transformation actions for the bottom-up approach are shown in the subsections of Section A.1 in Appendix A. They are the formalization of the transformation actions and form the set of transformation actions $P_{TA_{BU}}$ for the bottom-up approach.

$P_{ReuseAS}$ is a transformation action that allows one to reuse an already existing application service that realizes an information service modeled in a target business support map. Additionally, the application service to be reused is not allowed to be localized in the current business support map. $P_{ReuseAS}$ is applicable to transformation pattern (d). $P_{ReplaceAS}$ allows to consider a direct replacement of one application service through another. $P_{ReplaceAS}$ is applicable to transformation pattern (b). Using $P_{EnhanceAS_{BU}}$ takes an existing application service and enhances its functionality. $P_{EnhanceAS_{BU}}$ is applicable to transformation pattern (d).

The transformation actions $P_{NewASOldA}$ and $P_{NewASNewA}$ create new application services to provide the functionality. Figure 4.4 shows the LHS, RHS, and NAC of transformation action $P_{NewASOldA}$. The first part of the NAC of $P_{NewASOldA}$ (green dashed box and edges in Figure 4.4) prohibits a match of the transformation action if already an application service, that realizes the localized information service, is localized by the target business support element. Furthermore, the NAC, red box and edges in Figure 4.4, prohibits a match of $P_{NewASOldA}$ if an application service localized by a current business support element, that realizes the information service, localized by the target business support element, is present. Part of the LHS of $P_{NewASOldA}$ are all black and blue boxes and edges of Figure 4.4. Whereas, the blue 'localized by' edge only belongs to LHS and gets deleted, all black boxes and edges are also part of RHS. Besides, the black boxes and edges the RHS consists of the green boxes and edges of Figure 4.4. The RHS adds a node of type application service, its inherent edges, and additionally 'was localized by' edge for the information service.

$P_{NewASNewA}$ even creates a new application for implementing the new application service in contrast to $P_{NewASOldA}$ that uses an already existing application. $P_{NewASOldA}$ is applicable to transformation pattern (d) and $P_{NewASNewA}$ is applicable to transformation pattern (d).

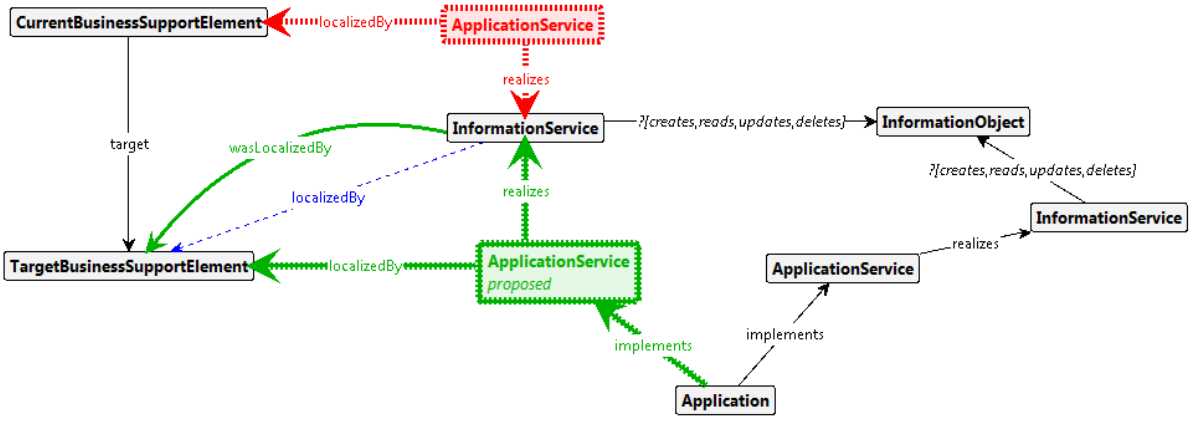

Figure 4.4: LHS, RHS, and NAC of Transformation Action $P_{NewASOldA}$

$P_{ReuseAS_{IS}}$ corresponds to the replacement of an existing application service, but without being specified in the first place, in contrast to $P_{ReplaceAS}$. The transformation actions $P_{EnhanceAS_{IS}}$, $P_{NewASOldA_{IS}}$, and $P_{NewASNewA_{IS}}$ are in the same way defined as the transformation actions without the IS suffix. The transformation with the IS suffix have two characteristics that differentiate them from the others:

1. RHS: the created or chosen application service becomes a successor of the application service currently in use

2. $P_{ReuseAS_{IS}}$, $P_{EnhanceAS_{IS}}$, $P_{NewASOldA_{IS}}$, and $P_{NewASNewA_{IS}}$ are applicable to transformation pattern (g), that means that there is already an application service in use

The transformation actions $P_{EnhanceAS_{BU}}$, $P_{NewASOldA}$, $P_{NewASNewA}$, $P_{EnhanceAS_{IS}}$, $P_{NewASOldA_{IS}}$, and $P_{NewASNewA_{IS}}$, are executable in normal, double, and quadruple resource mode. The LHS, NAC, and RHS are identical for the same transformation actions in different resource modes. However, they differ in the effort calculated for them. The transformation actions are formalized as typed attributed graph productions. An enterprise architect is able to select the applicable transformation actions generated through GG_{BU} during the process.

Initial State Bottom-Up Given the modeled business support maps for the bottom-up approach we create the initial state S_{BU} of GG_{BU} as a graph typed by PDM. Therefore, we need every application service and information service that is localized in the current and target business support maps. Furthermore, we make the following assumptions regarding information present in S_{BU}:

- For every application service, localized in a business support map, the application that implements it has been modeled

- For every information service, localized by a target business support element, the application services that realize it have been modeled

- For every information service the information objects that are created, read, updated, or deleted by it have been modeled

.

To be able to create the initial state we have to query the information from a knowledge base. Three SPARQL queries are necessary to retrieve the information from a knowledge base that uses semantic web technologies. The ontology uses German terms for the concepts and allows for a consideration of the English terms via labeling mechanisms.

For details on the ontology we refer to Appendix C.

Query 1 The first query retrieves all current business support elements and the application services localized by them for a certain business process. Furthermore, for the retrieved application services their implementing applications and the information services realized through them, including the information objects are queried. The listing below shows the corresponding SPARQL query that is applied on the EAM ontology (see Appendix C).

Listing 4.1: Query 1 for Initial State Bottom-Up

```
SELECT DISTINCT ?bbeIST ?prozessaktivitaet ?aservice ?iservice
                ?iobjekt ?anwendung
WHERE {
  ?bbeIST rdf:type lea:BebauungselementImIst.
  ?p lea:ProzessBestehtAusProzessaktivitaet ?prozessaktivitaet.
  ?bbeIST lea:BebauungselementBebautProzessaktivitaet ?prozessaktivitaet.
  OPTIONAL{
    ?bbeIST lea:BebauungselementImIstBebautAnwendungsservice ?aservice.
    ?aservice lea:AnwendungsserviceRealisiertInformationsservice ?iservice.
    ?anwendung lea:AnwendungImplementiertAnwendungsservice ?aservice.
    ?iservice lea:InformationsserviceVerwendetInformationsobjekt ?iobjekt.
  }
}
```

Query 2 The second query retrieves all target business support elements and the application services localized by them for a certain business process. Furthermore, for the retrieved application services their implementing applications and the information services realized through them, including the information objects are queried. The listing below shows the corresponding SPARQL query that is applied on the EAM ontology (see Appendix C).

Listing 4.2: Query 2 for Initial State Bottom-Up

```
SELECT DISTINCT ?bbeSOLL ?prozessaktivitaet ?aservice ?iservice
                ?iobjekt ?anwendung
WHERE {
  ?bbeSOLL rdf:type lea:BebauungselementImSoll.
  ?p lea:ProzessBestehtAusProzessaktivitaet ?prozessaktivitaet.
  ?bbeSOLL lea:BebauungselementBebautProzessaktivitaet ?prozessaktivitaet.
  OPTIONAL{
     ?bbeSOLL lea:BebauungselementImSollBebautAnwendungsservice ?aservice.
     ?aservice lea:AnwendungsserviceRealisiertInformationsservice ?iservice.
     ?anwendung lea:AnwendungImplementiertAnwendungsservice ?aservice.
     ?iservice lea:InformationsserviceVerwendetInformationsobjekt ?iobjekt.
  }
}
```

Query 3 The third query retrieves all information services that are localized within a target business support map for a certain process. Furthermore, all information objects

used by these information services are retrieved. The listing below shows the corresponding SPARQL query that is applied on the EAM ontology (see Appendix C).

Listing 4.3: Query 3 for Initial State Bottom-Up

```
1  SELECT DISTINCT ?bbeSOLL ?prozessaktivitaet ?iservice ?iobjekt
2  WHERE {
     ?bbeSOLL rdf:type lea:BebauungselementImSoll.
4    ?p lea:ProzessBestehtAusProzessaktivitaet ?prozessaktivitaet.
     ?bbeSOLL lea:BebauungselementBebautProzessaktivitaet ?prozessaktivitaet.
6    OPTIONAL{
        ?bbeSOLL lea:BebauungselementImSollBebautInformationsservice ?iservice.
8       ?iservice lea:InformationsserviceVerwendetInformationsobjekt ?iobjekt.
     }
10 }
```

Goal State Bottom-Up Informally the goal state for the bottom-up planning is defined as a state where every direct replacement of application services was applied and no more information service is localized by a target business support element. Figure 4.5 shows the nested typed attributed graph production to mark goal states within GG_{BU} via the typed attributed graph production $P_{goalState_{BU}}$ that creates a goal node.

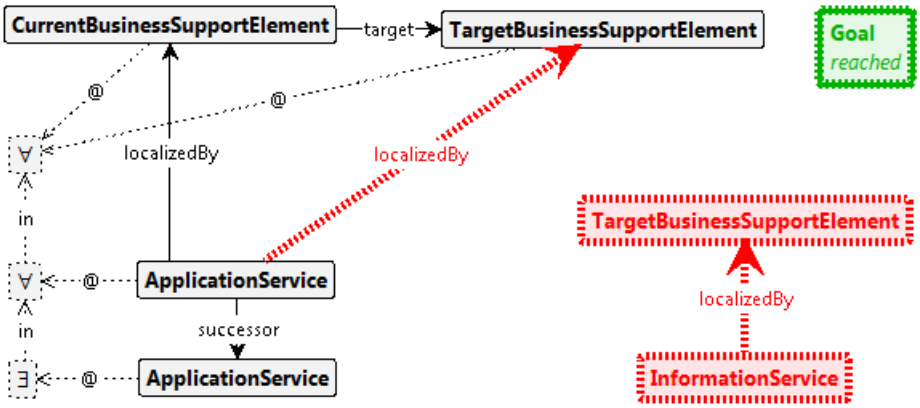

Figure 4.5: Nested Typed Attributed Graph Production $P_{goalState_{BU}}$ to Mark Goal States in Bottom-Up Approach

The NAC of $P_{goalState_{BU}}$ prohibits the application of the transformation action, if an information service is localized by a target business support element. Furthermore, part of the NAC is the red 'localized by' edge within the nesting. The nesting considers all current and target business support elements, that have an application service localized by a current but not by the target business support element. Only if a successor exists for those application services the LHS matches, i. e. all replacements have been considered in the target architecture. The RHS of $P_{goalState_{BU}}$ creates a goal node with the attribute value reached. All graphs G_i created through a sequence $S_{BU} \Rightarrow^* G_i$ of transformation actions that have a goal node form the set of goal states $G_{goal_{BU}}$.

4.2.2 Interactive Process for Bottom-Up Approach

Given the GG_{BU} we are now able to start the interactive decision support for target architecture selection by computing applicable transformation actions. Then the applicable transformation actions are ranked to provide the enterprise architect with an ordering on them. Afterwards, the enterprise architect selects a transformation action or starts a more sophisticated mechanism for selecting applicable transformation actions.

4.2.2.1 Computing Applicable Transformation Actions in Bottom-Up Approach

In the bottom-up approach no derivations from the description of commonalities are necessary besides the fact that we use S_{BU} to create the possible goal states. $P_{TA_{BU}}$, as the set of transformation actions available in the bottom-up approach, allows us to compute applicable transformation actions on S_{BU}, respectively one of its successor states.

4.2.2.2 Ranking Applicable Transformation Actions in Bottom-Up Approach

Below we give an informal description of how the efforts between the different transformation actions should be structured. The formal counterpart is given through Table 4.1, that shows the calculation scheme for the transformation actions in the bottom-up approach, and the concrete assignment of values through an enterprise architect.

$P_{ReuseAS}$ has the least effort, as it is not necessary to develop a new application service and there is currently no application service in use. Efforts may differ between applicable transformation actions as more employees may work in a role than in others. However, such information is not present in the states. We assume that the enterprise architect has this knowledge and considers them in the selection of applicable transformation actions. $P_{ReplaceAS}$ and $P_{ReuseAS_{IS}}$ create more effort than $P_{ReuseAS}$, as they include the change of the business support from an existing application service to another. Therefore, we double the fixed values for time and costs to take this replacement into account. However, no development efforts, that would result in variable costs, are involved. In contrast $P_{EnhanceAS_{BU}}$ involves development efforts but without changes from one application service to another. $P_{EnhanceAS_{IS}}$ creates a higher effort as it results in such a change from one existing application service to another.

The next highest effort is generated through $P_{NewASOldA}$ that creates an new application service using an existing application. Even though the development time is the same with $P_{EnhanceAS_{BU}}$ it creates higher costs through the introduction of a new application service that needs to be maintained. $P_{NewASOldA_{IS}}$ creates more effort as we double

the fixed costs to account for the organizational change from one application service to the new one. The second highest effort is assigned to $P_{NewASNewA}$, as it involves the creation of a new application. $P_{NewASNewA_{IS}}$ results in the highest effort as it includes a change in the business support. The different resource modes are considered in the ranking by using the scaling factors. An entry with '-' in Table 4.1 means that different resource modes are not available as only fixed values are involved in the calculation of the ranking.

4.2.2.3 Selecting Applicable Transformation Action in Bottom-Up Approach

After ranking the applicable transformation actions the enterprise architect chooses one and the state is changed according to the RHS. The chosen transformation action is added to $PD_{selected}$. Furthermore, we check if the state that results from the application of the transformation action is already in the set of explored states of GG_{BU}.

We have designed the transformation actions in such a way that it is not possible to reverse formerly applied transformation actions. This is realized by ensuring that every RHS of a transformation action is not reversible through the RHS of other transformation actions. The number of applicable transformation actions is limited by the number of transformation patterns (b), (d), and (g) present in S_{BU}. Furthermore, we need to assure that the graph transformation system created through the transformation actions is acyclic. We realize this by ensuring that parts of the RHS are also part of the NAC. As a consequence the application of a transformation action deletes its own LHS match in the graph. Therefore, we guarantee that the tree of graphs created through the application of transformation actions is acyclic.

The point in time for selecting a transformation action has no influence on its ranking. However, the selection of transformation action at a former point in time in the process may enable the selection of further transformation action alternatives. For example an applicable transformation action $P_{NewASOldA}$ may enable the execution of $P_{ReuseAS}$ which is not applicable in S_{BU}.

When $P_{goalState_{BU}}$ is applicable it is automatically applied and the enterprise architect is notified. She may now decide to further explore other goal states or proceed with the finalization of the goal state.

4.2.2.4 Finalizing a Selected Goal State in Bottom-Up Approach

Given the set of goal states $G_{goal_{BU}}$, we call the one chosen from the enterprise architect G_{goal}. Given this state we finalize it to create a complete EAG_{target}.

Resource Mode	Transformation Actions				
	$R_{P_{ReplaceAS}}$	$R_{P_{ReuseAS}}$	$R_{P_{EnhanceAS_{BU}}}$	$R_{P_{NewASOldA}}$	$R_{P_{NewASNewA}}$
Normal	$(C_f \times 2)^{w_1} \times$ $(T_f \times 2)^{w_2} \times$ $(Z_{P_{ReplaceAS}})^{w_3}$	$(C_f)^{w_1} \times$ $(T_f)^{w_2} \times$ $(Z_{P_{ReuseAS}})^{w_3}$	$(C_v + C_f)^{w_1} \times$ $(T_f + T_v)^{w_2} \times$ $(Z_{P_{EnhanceAS_{BU}}})^{w_3}$	$(C_v + C_f + C_s)^{w_1} \times$ $(T_f + T_v)^{w_2} \times$ $(Z_{P_{NewASOldA}})^{w_3}$	$(C_v + C_f + C_a)^{w_1} \times$ $(T_f + T_v)^{w_2} \times$ $(Z_{P_{NewASNewA}})^{w_3}$
Double	-	-	$((c_d \times C_v) + C_f)^{w_1} \times$ $(T_f + (t_d \times T_v))^{w_2} \times$ $(Z_{P_{EnhanceAS_{BU}}})^{w_3}$	$((c_d \times C_v) + C_f + C_s)^{w_1} \times$ $(T_f + (t_d \times T_v))^{w_2} \times$ $(Z_{P_{NewASOldA}})^{w_3}$	$((c_d \times C_v) + C_f + C_a)^{w_1} \times$ $(T_f + (t_d \times T_v))^{w_2} \times$ $(Z_{P_{NewASNewA}})^{w_3}$
Quadruple	-	-	$((c_q \times C_v) + C_f)^{w_1} \times$ $(T_f + (t_q \times T_v))^{w_2} \times$ $(Z_{P_{EnhanceAS_{BU}}})^{w_3}$	$((c_q \times C_v) + C_f + C_s)^{w_1} \times$ $(T_f + (t_q \times T_v))^{w_2} \times$ $(Z_{P_{NewASOldA}})^{w_3}$	$((c_q \times C_v) + C_f + C_a)^{w_1} \times$ $(T_f + (t_q \times T_v))^{w_2} \times$ $(Z_{P_{NewASNewA}})^{w_3}$
Resource Mode	Transformation Actions continued				
	$R_{P_{ReuseAS_{IS}}}$	$R_{P_{EnhanceAS_{IS}}}$	$R_{P_{NewASOldA_{IS}}}$	$R_{P_{NewASNewA_{IS}}}$	
Normal	$(C_f \times 2)^{w_1} \times$ $(T_f \times 2)^{w_2} \times$ $(Z_{P_{ReuseAS_{IS}}})^{w_3}$	$((C_f \times 2) + C_v)^{w_1} \times$ $((T_f \times 2) + T_v)^{w_2} \times$ $(Z_{P_{EnhanceAS_{IS}}})^{w_3}$	$((C_f \times 2) + C_v + C_s)^{w_1} \times$ $((T_f \times 2) + T_v)^{w_2} \times$ $(Z_{P_{NewASOldA_{IS}}})^{w_3}$	$((C_f \times 2) + C_v + C_a)^{w_1} \times$ $((T_f \times 2) + T_v)^{w_2} \times$ $(Z_{P_{NewASNewA_{IS}}})^{w_3}$	
Double	-	$((C_f \times 2) + (c_d \times C_v))^{w_1} \times$ $((T_f \times 2) + (t_d \times T_v))^{w_2} \times$ $(Z_{P_{EnhanceAS_{IS}}})^{w_3}$	$((C_f \times 2) + (c_d \times C_v) + C_s)^{w_1} \times$ $((T_f \times 2) + (t_d \times T_v))^{w_2} \times$ $(Z_{P_{NewASOldA_{IS}}})^{w_3}$	$((C_f \times 2) + (c_d \times C_v) + C_a)^{w_1} \times$ $((T_f \times 2) + (t_d \times T_v))^{w_2} \times$ $(Z_{P_{NewASNewA_{IS}}})^{w_3}$	
Quadruple	-	$((C_f \times 2) + (c_q \times C_v))^{w_1} \times$ $((T_f \times 2) + (t_q \times T_v))^{w_2} \times$ $(Z_{P_{EnhanceAS_{IS}}})^{w_3}$	$((C_f \times 2) + (c_q \times C_v) + C_s)^{w_1} \times$ $((T_f \times 2) + (t_q \times T_v))^{w_2} \times$ $(Z_{P_{NewASOldA_{IS}}})^{w_3}$	$((C_f \times 2) + (c_q \times C_v) + C_a)^{w_1} \times$ $((T_f \times 2) + (t_q \times T_v))^{w_2} \times$ $(Z_{P_{NewASNewA_{IS}}})^{w_3}$	

Table 4.1: Efforts of Transformation Actions in Bottom-Up Approach

Therefore, we take G_{goal} as the initial state $S_{finalize}$ for $GG_{finalize}$. We argue that the following properties for $S_{finalize}$ hold:

- every newly created application service and application have the attribute proposed
- every functional enhanced application service has the attribute to be extended
- for every transformation pattern (b) and (g) that was present in S_{BU}, a successor edge was created from the application service localized in the current business support map to the application service localized in the target business support map

This is a consequence of the design of the transformation actions where we ensure these properties through the RHS of the transformation actions. We introduce typed attributed graph transformations for $GG_{finalize}$. They allow us to finalize the states needed for the transformation path generation.

We introduce attributed typed graph productions to finalize $G_{goal_{BU}}$ and number them from one to ten to express their ordering: $P_{finalize_{BU1}} \dots P_{finalize_{BU10}}$. Their LHS, NAC, and RHS are depicted in the subsections B.1 to B.10.

$P_{finalize_{BU1}}$ adds the attribute stable to all application services that are not in the lifecycle phase proposed and are localized by a target business support element. Furthermore, we add to all application services that are not functional enhanced or proposed, and not localized by a current business support element the attribute stable via $P_{finalize_{BU2}}$. With $P_{finalize_{BU3}}$ we set all remaining application services to the lifecycle phase live. Now every application service is either stable, live or proposed and may be additionally tagged for extension through functional enhancement.

$P_{finalize_{BU4}}$ and $P_{finalize_{BU5}}$ add the lifecycle phases to the applications. All applications that are not proposed and implement an application service which is proposed or stable are set to stable ($P_{finalize_{BU4}}$). All other applications, that are neither proposed nor stable, are set to live ($P_{finalize_{BU5}}$).

Through the application of transformation actions cyclic successor dependencies may have been created. We remove them through $P_{finalize_{BU6}}$ and $P_{finalize_{BU7}}$. At first we remove the last successor relationship of a transitive chain of successor relationships and add one from the second last to the first element of the chain ($P_{finalize_{BU6}}$). This is done until the successor relationships become direct cyclic, i.e. application service AS1 is successor of AS2 and AS2 is successor of AS1. Direct cyclic relationships are removed through ($P_{finalize_{BU7}}$). Through $P_{finalize_{BU8}}$ we add the successor relationships for the applications. Through $P_{finalize_{BU9}}$ and $P_{finalize_{BU10}}$ we remove transitive cyclic and direct cyclic successor relationships in the same way as for the applications services.

4.2.3 Output of the Bottom-Up Approach

As a result we created the following information in the bottom-up approach:

1. $PD_{selected}$ with a ranking for the different transformation actions selected, including the different values for the dimensions cost, time, and risk

2. EAG_{target} neglecting the usage dependencies between applications and application services

3. $transformationModel$ for applications and application services in $EAG_{current}$ and EAG_{target}

The enterprise architect may now proceed with finalizing the target architecture by changing the dependencies between applications and application services (see Section 4.4).

4.3 Top-Down Approach

We introduce the top-down approach starting with the necessary input. Then we present the process for the top-down approach, followed by the output of the process. The top-down approach consists of two phases. Within the first phase decision making upon applications supported. Within the second phase the detailed decision making on application services is supported.

4.3.1 Input for Top-Down Approach

We now describe the input that we need from an enterprise architect to be able to start the interactive interactive decision support for target architecture selection using a top-down approach and the formalisms to create GG_{TD}.

4.3.1.1 Input from Enterprise Architect

As an input from the enterprise architect we need the modeled current and target business support maps on a coarse level. Furthermore, it is necessary to retrieve the weights, scaling factors for the resource modes, and the risk value to allow for a ranking of the applicable transformation actions.

Top-Down Business Support Map Figure 4.6 shows an example for a current and a target business support map in the top-down approach. We refer to subsection 7.3 for further details on the business support map and its context.

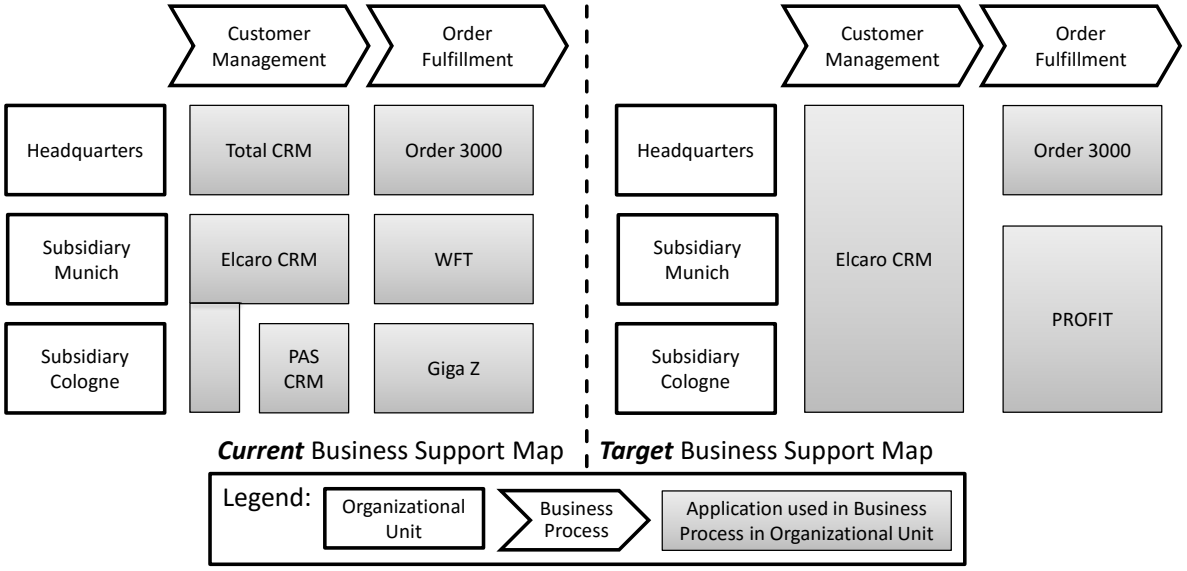

Figure 4.6: Example of Current and Target Business Support Map on a Value Chain Level for Top-Down Approach

Entries in the cells of the business support map are either:

- not possible; as the organizational unit may not be involved in the business process
- empty; no business support of a business process through an application
- an application
- a logical application in the target business support map

Modeling the current and target business support maps takes place in the EA tool used by the enterprise architect. The modeled current and target business support maps, including the implicit information as mentioned in subsection 4.1.1, are then exported and transformed into the initial state S_{TD}.

4.3.1.2 Top-Down Formalisms

In the following we introduce the initial state and the nested typed attributed graph production to detect goal states.

Transformation Patterns Top-Down Figure 4.7 shows the transformation patterns we assign a meaning to in the top-down approach and one ambiguous situation.

Transformation pattern (i) is an abstraction of a situation where one or more applications are to be replaced by a certain application. (ii) is used to express that the business support is to be delivered through the same application in the target enterprise architecture as in the current. In contrast transformation pattern (v) states that there is currently no business support and no support through an application should be planned. Transformation pattern (iii) means that there is currently a business support through an application but in the future there should be no support through an application. (iv) in contrast is the abstraction of a situation where currently no business support through an application takes place, but in the target architecture there should be a support.

Situations where currently no business support exists and a logical application is modeled in the target business support map. In contrast transformation pattern (vii) has a logical application component modeled in the target business support map, but one or more applications are localized by the corresponding current business support elements. Figure 4.7 also shows the ambiguous situation where it not clear how this bidirectional replacement should be considered in transformation paths.

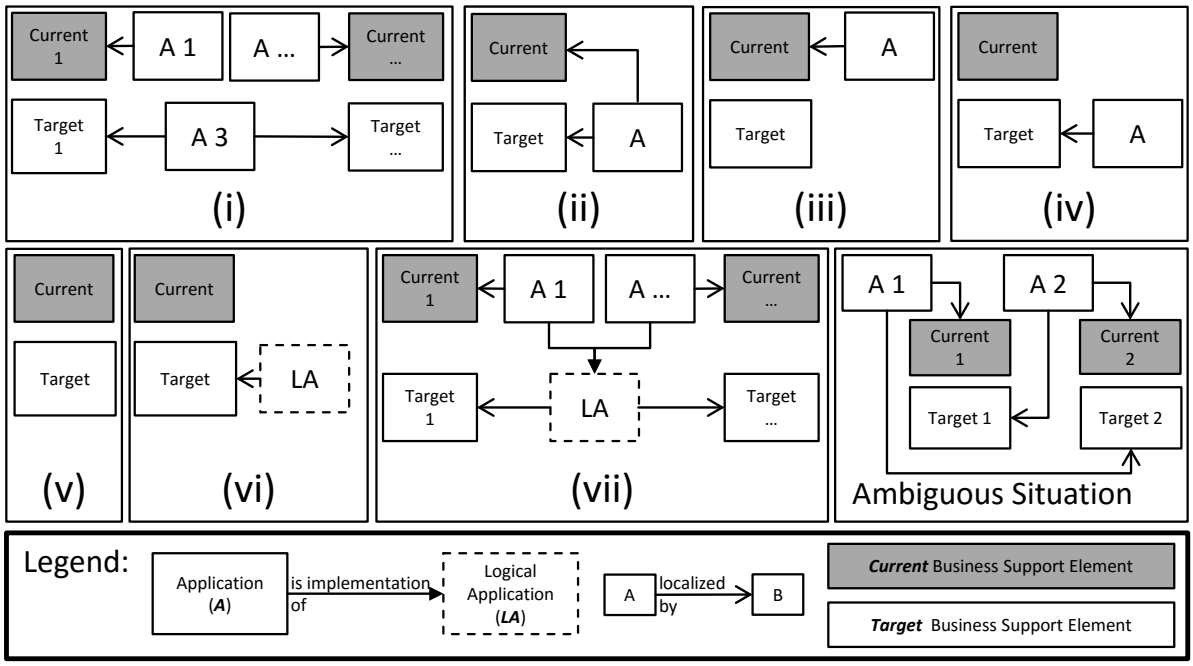

Figure 4.7: Transformation Patterns in Top-Down Approach

Please note that we will not provide transformation actions for the transformation patterns (iii), (iv), and (vi), because we cannot derive existing detailed information that would allow us to provide a ranking or any meaningful suggestions.

Transformation Actions Top-Down The set of transformation actions $P_{TA_{TD}}$ for the top-down approach consists of the transformation actions for the first and second phase. Their LHS, NAC, and RHS are shown in section A.2.

The transformation actions introduced for top-down planning in the first phase are:

- $P_{ReuseApp}$
- $P_{ReplaceApp}$
- P_{NewApp}

$P_{ReplaceApp}$ is applicable to transformation pattern (i). It makes the application in the target business support map a successor of the applications specified for the same business support elements in the current business support map. $P_{ReuseApp}$ and P_{NewApp} are applicable to transformation pattern (vii). The former transformation action allows to reuse an existing application that is already an implementation of the logical application. However, it is not allowed that one of the applications localized by one of the current business support elements is reused. Via P_{NewApp} a proposed application as a new implementation of the logical application is introduced. Figure 4.8 shows the LHS, RHS, and NAC of transformation action P_{NewApp}.

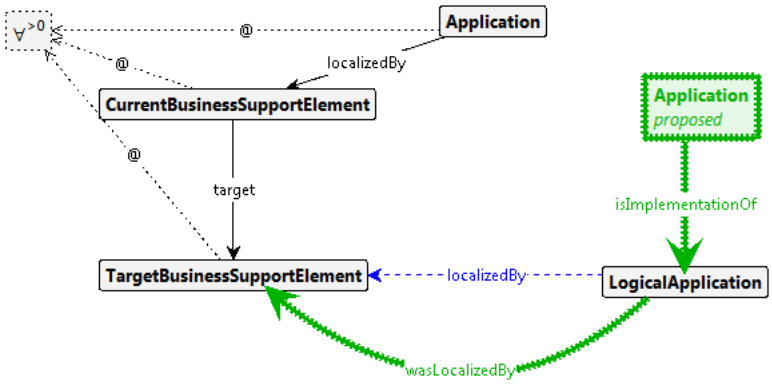

Figure 4.8: LHS, RHS, and NAC of Transformation Action P_{NewApp}

For P_{NewApp} the typed attributed graph production creates for every target business support element, the logical application is localized at, a new graph node of type application. Furthermore, each of these applications becomes an implementation of the logical application component and the logical application is no longer localized by the target business support element. However, the intended effect is that only one application is present in the target architecture. Therefore, we automatically merge the created application nodes in the background by applying the typed attributed graph production $P_{mergeNewApps}$ as shown in Figure B.11.

Regardless of whether $P_{ReuseApp}$ or P_{NewApp} is selected the application that becomes localized by the target business support elements is a successor of the ones localized by the current business support elements.

The transformation actions introduced for top-down planning in the second phase are:

- $P_{EnhanceAS_{TD}}$
- $P_{NewAS_{TD}}$

$P_{EnhanceAS_{TD}}$ allows to enhance an application service for existing or proposed applications. An application service for a proposed application is only enhanceable when $P_{NewAS_{TD}}$ was selected at least one time before, because the LHS of $P_{EnhanceAS_{TD}}$ requires an already existing application service to match. $P_{NewAS_{TD}}$ develops a new application service for an existing or proposed application.

Initial State Top-Down With the modeled business support maps for the top-down approach at hand we create the initial state S_{TD} of GG_{TD}. S_{TD} is a graph typed by PDM. Therefore, we need every application and logical application that is localized in the current and target business support maps. Furthermore, we make the following assumptions regarding information present in S_{TD}:

- For every application, localized in a business support map, the application services it implements have been modeled
- For every logical application, localized by a target business support element, the applications that are an implementation of it have been modeled
- For every application service the information service they realize and their respective information objects have been modeled

The SPARQL queries that retrieve the necessary information for the initial state in the top-down approach are similar to the ones in bottom-up approach. The main difference is that we have to query the applications and logical applications localized within the current and target business support maps. Furthermore, we have to query the application services, information services and information objects that are directly or indirectly related to these applications. We do not provide the SPARQL queries here, due to the similarity with the bottom-up queries.

Goal State Top-Down The goal state for the first phase of top-down planning is defined as a state where no logical application is localized by a target business support.

Furthermore, for every occurrence of an application localized by a current business support element, but not by the corresponding target business support element, a successor application for the target business support elements is specified. All graphs G_i created, through a sequence $S_{TD} \overset{p,m}{\Longrightarrow}^* G_i$ of transformation actions, are states that can be refined in the second phase. Figure 4.9 shows the nested typed attributed graph production to mark those states within GG_{TD} via the typed attributed graph production $P_{intermediateGoalState_{TD}}$.

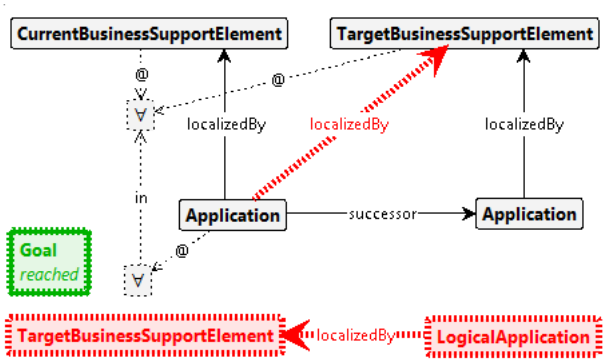

Figure 4.9: Nested Typed Attributed Graph Production $P_{intermediateGoalState_{TD}}$ to Mark Goal States in Top-Down Approach

The NAC of $P_{intermediateGoalState_{TD}}$ consists of the red and green dashed boxes and edges in Figure 4.9. It is neither allowed that already a goal node is present nor that a logical application is localized by a target business support element. The former condition prohibits the recursive application of $P_{intermediateGoalState_{TD}}$, whereas the latter condition ensures that for every logical application an application has been determined. Furthermore, part of the NAC is involved in the nesting of Figure 4.9 with the $\forall$-operators. Therefore, only applications not localized by a target business support element are considered in the nesting for every current and target business support element. This corresponds to the different levels of nesting in $P_{intermediateGoalState_{TD}}$, whereas the current and target business support elements are on the highest nesting level and the application with a successor on the lowest. The selected goal state gets refined in the second phase.

The goal state for the second phase of top-down planning is neither defined explicitly as a state nor as a state that has to satisfy certain criteria. This is due to the fact that there are situations in the top-down approach in which transformation action might not be applied even though one could be applied. The enterprise architect might leave the transformation action out, by not selecting it. She then marks the state as a goal state by selecting $P_{goalState_{TD}}$. If she selects all applicable transformation actions and no other is available $P_{goalState_{TD}}$ gets selected automatically. Before the second phase starts the goal node from the first phase is removed, thereby it can be created by $P_{goalState_{TD}}$ again to mark the goal states in the second phase.

4.3.2 Interactive Process for Top-Down Approach

Given the GG_{TD} we are now able to start the interactive decision support for target architecture selection by computing applicable transformation actions. Then the applicable transformation actions are ranked to provide the enterprise architect with an ordering on them. Afterwards, the enterprise architect selects a transformation action or starts a more sophisticated mechanism for selecting applicable transformation actions. The interactive process for the top-down approach is divided in two phases, whereas the second phase consists of the same steps as the first. However, different goal states and transformation actions are considered.

4.3.2.1 Computing Applicable Transformation Actions in Top-Down Approach

Besides, using S_{TD} to create the possible goal states there are no derivations from the description of commonalities. $P_{TA_{TD}}$, as the set of transformation actions available in the top-down approach, allows us to compute applicable transformation actions on S_{TD}, respectively one of its successor states.

4.3.2.2 Ranking Applicable Transformation Actions in Top-Down Approach

Based on the definition of functional deltas, we calculate the ranking for the transformation actions. The formal part of the calculation is given through Table 4.2, that shows the calculation scheme for the transformation actions of the first phase in the top-down approach.

Definition 28. *Functional Delta: The functional delta between application A1 and A2 are the information services not realized through application services of application A2 through application services from A1. The functional delta between A2 and A1 is defined vice versa. Two applications with two empty functional deltas implement application services that realize the same information services.*

Depending on the functional delta between the applications localized by the current business support elements and the application localized by the target business support element the effort varies. $P_{ReplaceApp}$ and $P_{ReuseApp}$ result in the same effort if the functional delta between the explicitly stated replacement (transformation pattern (i)) is equivalent to the functional delta between the application suggested for reuse. P_{NewApp} results in higher efforts as a new application and the application services to cover the functional delta need to be implemented .

Definition 29. *Aggregated Functional Delta: The aggregated functional delta between an application A1 and at least two other applications are the information services that are not realized through A1's application services but through any of the other applications. However, each of the information services is only considered once for A1. The aggregated functional delta has no inverse. An application A1 with an empty aggregated functional delta to several other applications implements all the functionality of them.*

The aggregated functional delta allows to calculate the efforts for transformation patterns where more than two applications are involved in a more compact way. Calculating the ranking of transformation actions that create new applications in particular benefit from the derivation of an aggregated functional delta, as the new application needs to realize all information services of the applications it is intended to replace, but only once.

The transformation actions $P_{ReuseApp}$, $P_{ReplaceApp}$, and P_{NewApp} are not executable in different resource modes. We argue that the development of an application in different resource modes is just a development of its functionality in different resource modes. Therefore, we provide a Min, Max frame for the ranking that allows us to give the enterprise architect a lead what the minimum and maximum efforts are. This frame is given through the functional delta. The detailed decisions are provided afterwards, considering different resource modes for the subsequent transformation actions that allow to calculate efforts in detail.

	$R_{P_{ReplaceApp}}$	$R_{P_{ReuseApp}}$	$R_{P_{NewApp}}$
Min	$(C_f + C_{v_{min}})^{w_1} \times$ $(T_f + T_{v_{min}})^{w_2} \times$ $(Z_{ReplaceApp_{min}})^{w_3}$	$(C_f + C_{v_{min}})^{w_1} \times$ $(T_f + T_{v_{min}})^{w_2} \times$ $(Z_{ReuseApp_{min}})^{w_3}$	$(C_f + C_{v_{min}} + C_a)^{w_1} \times$ $(T_f + T_{v_{min}})^{w_2} \times$ $(Z_{NewApp_{min}})^{w_3}$
Max	$(C_f + C_{v_{max}})^{w_1} \times$ $(T_f + T_{v_{max}})^{w_2} \times$ $(Z_{ReplaceApp_{max}})^{w_3}$	$(C_f + C_{v_{max}})^{w_1} \times$ $(T_f + T_{v_{max}})^{w_2} \times$ $(Z_{ReuseApp_{max}})^{w_3}$	$(C_f + C_{v_{max}} + C_a)^{w_1} \times$ $(T_f + T_{v_{max}})^{w_2} \times$ $(Z_{NewApp_{max}})^{w_3}$

Table 4.2: Efforts of Transformation Actions for Applications in Top-Down Approach

C_v and T_v are dependent on the functional delta and if P_{NewApp} is selected additionally the efforts for the development of that new application need to be considered. $C_{v_{min}}$ is calculated by assuming that as much application services as possible are extended in the normal resource mode. $C_{v_{max}}$ assumes that all functional deltas are realized through new application services in quadruple resource mode. For $T_{v_{min}}$ and $T_{v_{max}}$ holds the same as for the costs regarding the minimum and maximum realization time. For the risk of the transformation actions we provide a minimum and maximum, too.

Risk depends also on the functional delta and assume only enhancements, as far as possible, for the minimum. For the maximum of the risk development of new application services are considered in the risk. Therefore, we are able to provide a frame, whereas the detailed decisions regardless of their employed resource modes are within the Min, Max frame.

Please note that the calculation scheme of the ranking for the transformation actions $R_{P_{ReplaceApp}}$ and $P_{ReuseApp}$ is the same. This is because applying $P_{ReuseApp}$ results in the same efforts as if the enterprise architect would have specified an explicit replacement via transformation pattern (i).

4.3.2.3 Selecting Applicable Transformation Action in Top-Down Approach

After ranking the applicable transformation actions the enterprise architect chooses one and it is added to $PD_{selected}$. Furthermore, the graph is changed according to the RHS of the transformation action.

For every G_i created through selecting an applicable transformation action it needs to be checked if G_i is already present in the set of graphs created through a sequence of applied transformation actions $S_{TD} \overset{p,m}{\Longrightarrow}^{*} G_j$. Therefore, we need to check if there exists an isomorphism from G_i to G_j.

$P_{intermediateGoalState_{TD}}$ is automatically applied if it is applicable. The enterprise architect is notified that the first phase of the interactive decision support for target architecture selection is finished. She may explore states that result from the selection of other transformation actions or proceed with the second phase. If she proceeds with the second phase the selected state, resulting from $P_{intermediateGoalState_{TD}}$ is called $G_{intermediate}$.

During the first phase successor relationships between applications were set. Every application present in $G_{intermediate}$ belongs to exactly one set of successor bundles (see Figure 3.2). Depending on the set an application belongs to we create different suggestions for the enterprise architect.

We now take care of the functional deltas between the applications to be shutdown and to be developed or already existing applications. Therefore, we need transformation actions that take care of the functional deltas. In the second phase only the transformation actions $P_{EnhanceAS_{TD}}$ and $P_{NewAS_{TD}}$ are available for selection. Their applicability is checked on $G_{intermediate}$, respectively one of its successor states G_j, given through a sequence of transformation actions $G_{intermediate} \overset{p,m}{\Longrightarrow}^{*} G_j$. Each of the suggestions can be ranked according to the ranking as introduced in Table 4.1, with $R_{P_{EnhanceAS_{TD}}} = R_{P_{EnhanceAS_{BU}}}$ and $R_{P_{NewAS_{TD}}} = P_{NewASOldA}$ and its respective resource modes. The suggestions themselves were already introduced in Diefenthaler and Bauer (2013).

1. *noSuccessor* set: for each implemented application service in the current architecture check if it is used by an application that is part of the target architecture or if the application has a successor relationship.

   a) If there are any applications it is necessary to check if they still can work properly without using the application service.

   b) Otherwise, no information from the current architecture needs to be added to the target architecture.

2. *noPredecessor* set: it is not possible to suggest a detail for the target architecture as there exists no detail in the current architecture. A manual addition of implemented application services, information services and their information objects in the target architecture is necessary.

3. *oneToMany* set:

   a) If the predecessor is part of *onlyCurrent* all provided application services of the predecessor, including their information services, are suggested to be implemented by one of the successor applications.

   b) Otherwise, all implemented application services and information services of the predecessor are suggested to be provided by one of the successor applications or the remaining part of the predecessor in the target architecture.

4. *manyToOne* set:

   a) If the successor is part of *onlyTarget* it is suggested to implement each application service of its successors, but only one per information service.

   b) Otherwise, it is suggested that the successor implements the application services already provided in the current architecture, i. e. by itself, and provide all application services of the other predecessors, but only one per information service.

5. *manyToMany* set: All implemented application services are suggested to be implemented by one of the successors. If more than one predecessor implements an application service with the same information service the suggestion is to provide only one application service in the target architecture with such an information service.

6. *oneToOne* set: all application services, including their information services, provided by the predecessor are suggested to be provided by the successor.

The interactive decision support for target architecture selection is considered as finished as soon as $P_{goalState_{TD}}$ is applicable and the enterprise architect is notified. She may proceed with adding or enhancing application services, via $P_{EnhanceAS_{TD}}$ and $P_{NewAS_{TD}}$.

Each of these suggestions, if applied, changes $G_{intermediate}$. After the changes, the enterprise architect wants to consider, $G_{intermediate}$ has become EAG_{target}, without the usage dependencies between applications and application services.

4.3.2.4 Finalizing a Selected State in Top-Down Approach

The following properties for the state, in which $P_{intermediateGoalState_{TD}}$ is applicable, hold:

- every new introduced application has the attribute proposed
- every reused application is now localized by the appropriate target business support elements
- for every transformation pattern (i) and (vii) that was present in S_{TD}, a successor edge was created from the application localized in the current business support map to the application localized in the target business support map, as a result it also contains $transformationModel$ for the applications

For the state in which $P_{goalState_{TD}}$ is applicable the following properties hold:

- every new introduced application service has the attribute proposed
- every functional enhanced application service has the attribute to be extended
- every application that does not have a successor does not implement an application service which has an successor

For each application service information is stored if it is the successor of one or more application services in the current architecture. This is necessary to create the transformation model for the application services, that in turn allows a sound sequencing of the transformation actions later on.

4.3.3 Output of the Top-Down Approach

As a result we created the following information in top-down approach:

1. EAG_{target} neglecting the usage dependencies between applications and application services

2. $PD_{selected}$ with a ranking for the different transformation actions selected, including the different values for the dimensions cost, time, and risk

3. $transformationModel$ for applications and application services in $EAG_{current}$ and EAG_{target}

The enterprise architect may now proceed with finalizing the target architecture by changing the dependencies between applications and application services (see Section 4.4).

4.4 Considering Application Service Dependencies in EAG_{target} and Manual Changes

We have not considered the usage dependencies between applications and application services. As we want to create a target enterprise architecture EAG_{target} on the same level of detail as $EAG_{current}$, we need to consider this information in the transformation path generation. Therefore, we add the dependencies and change them as suggested below. We call these changes suggestions as it is not possible to derive with the information at hand all dependencies that may be created or deleted through an enterprise architect.

Each application belongs to one of the following successor bundle types: *manyToMany*, *oneToOne*, *manyToOne*, *oneToMany*, *noPredecessor*, and *noSuccessor*. In the cases below we do not differentiate between the extended successor relationship bundles as introduced in Figure 3.2 in chapter 3.

The suggestions for the changes regarding the usage dependencies between applications and application services were already presented in Diefenthaler and Bauer (2013). They are as follows:

1. *manyToMany* set: all used application services of predecessors are suggested to be used by at least one successor. The user can choose if more than one successor should use the application service of a predecessor.

2. *oneToOne* set: all application services used by the predecessor are suggested to be used by the successor.

3. *manyToOne* set: used application services of the predecessors are suggested to be also used in the target enterprise architecture.

4. *oneToMany* set:

   a) If the predecessor is part of *onlyCurrent* all used application services of the predecessor are suggested to be used by one of the successor applications.

   b) Otherwise, all used application services of the predecessor are suggested to be used by one of the successor applications or the remaining part of the predecessor in the target architecture.

5. *noPredecessor* set: which application services are used by the application needs to be modeled manually as no information from the current architecture is available.

6. *noSuccessor* set: as the application does not exist in the target enterprise architecture no information about used application services needs to be added to the target enterprise architecture.

Even though we formalized these suggestions as typed attributed graph productions we cannot determine whether the dependencies are to be present in the target architecture finally. This knowledge is implicit to the enterprise architect and in case it is not she has to ask the owner of the applications whose dependencies are unclear in the target architecture. Therefore, we do not call the changes of usage dependencies transformation actions.

After the dependencies are changed and no application service that belongs to *onlyCurrent* is used by any application we have finalized EAG_{target} including the usage dependencies between applications and application services in the target architecture.

The enterprise architect may now model additional provided application services or let a suggested application service be implemented by an application that is not a successor of the application that provided it in the current architecture. Furthermore, she may add or remove usage dependencies or functional enhance or decrement application services.

When the target architecture is considered as completed it is possible to conduct a gap analysis on the complete current and target architecture and retrieves the sets $onlyCurrent$, $stable$, and $onlyTarget$. This allows the enterprise architect to gain a quick overview on elements, relationships, and attributes that are to be changed in the transformation but also other changes that were modeled. The mechanism behind the gap analysis is to check every modeled information in the current architecture for presence in the target architecture. Afterwards, the same is done vice versa. This allows to determine whether parts of current architecture are only present in it. Such a mechanism is often termed as a *diff*-tool, as it shows differences between two information sources.

Kremen et al. (2011) provide such a tool for OWL ontologies that would even allow to compare changes in the semantics of an ontology, i. e. changes in the terminological box.

For further technical details we refer to Diefenthaler and Bauer (2013). The approach presented in Diefenthaler and Bauer (2013) also describes how the gap analysis may be already applied after the first phase of the top-down approach.

4.5 Summary of Interactive Decision Support for Target Architecture Selection

In the following chapter we presented our approach for interactive decision support for target architecture selection with business support maps. Therefore, we introduced a bottom-up approach and a top-down approach with their commonalities and differences, for example in creating a ranking. We described necessary input from an enterprise architect for both approaches and formalized the underlying models. Additionally, we showed how the Weighted Product Model is used as a multi-criteria decision making technique to rank the transformation actions. Furthermore, we showed how transformation actions are computed and can be applied in different resource modes. Lastly, we pointed out how the target architecture is finalized regarding the usage dependencies between applications and application services.

5 Interactive Transformation Path Generation

We present our approach for the transformation path generation, that is concerned with the actual sequence in which the changes are to be realized. At first we introduce the necessary input from an enterprise architect to start the interactive process. Furthermore, we introduce the necessary formalisms that allow us to generate transformation paths from a current to a target architecture. We also introduce a formalism to consider preferred detours in the transformation paths. Afterwards, we describe the process for computing and selecting a sequence of transformation actions to generate the transformation path. We end the chapter with an overview of the output of the interactive selection process and a summary.

5.1 Input for Interactive Transformation Path Generation

In this section we describe the input we need from an enterprise architect and the underlying formalisms to start the interactive process of transformation path generation.

5.1.1 Input from Enterprise Architect

Our assumption regarding reusable models is that an enterprise architect has either modeled manually a transformation model, and its current and target enterprise architecture or used the approach described in Chapter 4. Furthermore, we assume that information regarding the transformation actions' cost, time, and risk is already available in $PD_{selected}$. If they are not available we would have to calculate them in order to provide their duration, cost, and risk as described in the Subsubsections 4.2.2.2 and 4.3.2.2.

5.1.2 Formalisms

We use the graph grammar GG_{TP} for the generation of the transformation paths. GG_{TP} and its necessary components are defined as follows:

- GG_{TP} is the typed graph grammar used for transformation path generation consisting of GTS_{TP} and S_{TP}
- GTS_{TP} as a typed attributed graph transformation system consisting of $DSIG_{PDM}$, PDM and P_{TP}
- $DSIG_{PDM}$ as defined in Subsection 3.5.4
- PDM as defined in Subsection 3.5.3
- $P_{TP} = P_{TP_{TA}} \cup P_{goalState_{TP}} \cup P_{TP_{unique}}$ with:
  - $P_{TP_{TA}}$ as the transformation actions for transformation path generation (see subsubsection 5.1.2.2)
  - $P_{goalState_{TP}}$ as the typed attributed graph production to detect the goal state
  - $P_{TP_{Unique}}$ as the typed attributed graph production that are bound to individuals and take care of unique dependencies (see subsubsection 5.1.2.3) between applications and application services
- S_{TP} as the initial state of GG_{TP}

5.1.2.1 Transformation Patterns in Transformation Path Generation

We support the extended sets of successor bundles for applications and application services which are an specialization of Figure 3.2. The supported transformation patterns are shown in Figures 5.1 and 5.2. We consider the different successor bundles for applications and application services: *manyToMany*, *oneToOne*, *manyToOne*, *oneToMany*, *noPredecessor*, and *noSuccessor*. Furthermore, we allow for the same applications and application services to be present in the successor bundles *manyToMany*, *oneToOne*, *manyToOne*, and *oneToMany*.

Combining the transformation patterns is not limited except to the ambiguous situations, we will describe in the following. Figure 5.3 shows these ambiguous situations that are not considered in transformation path generation.

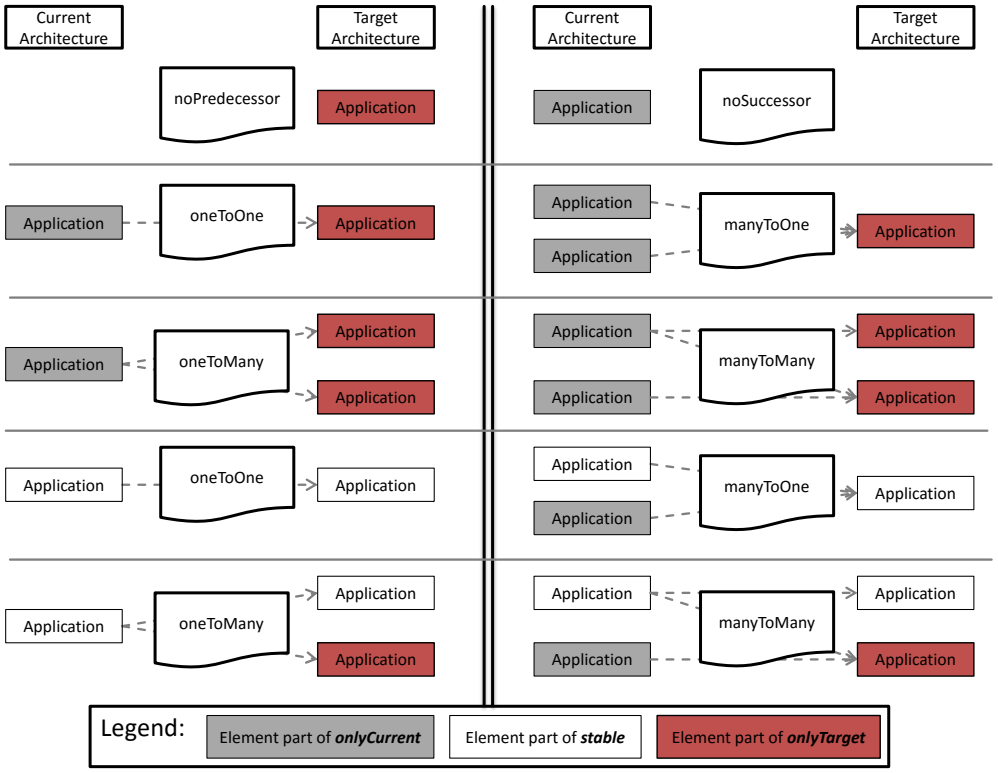

Figure 5.1: Transformation Patterns for Applications

The ambiguity in the situations is that it is unclear how the situation has to be interpreted. An application that is to be shutdown cannot implement an application service that is to be present in the target architecture, as the application is no longer in operation (Figure 5.3, (i) and (ii)). A similar ambiguous situation is present when an application that is to be developed implements an application service that either belongs to *onlyCurrent* or belongs to *stable* (Figure 5.3, (iii) and (iv)). In both cases the application service is not usable in the current architecture, however it is live in that state.

There are also ambiguous situations that are not allowed to be present for usage dependencies between applications and application services. Figure 5.3 (v) and (vi) show an application that belongs to *onlyCurrent* that uses either an application service that belongs to *stable* or *onlyTarget*. In both cases the usage dependency is meaningless, if present in the target architecture, as a application that is not in operation cannot have dependencies to application services. Furthermore, an ambiguous situation is present if an application that belongs to *onlyTarget* uses an application in the current architecture (Figure 5.3, (vii) and (viii)). This is not possible as an application that is not in operation cannot have dependencies before it is live. Please note that the restrictions for the usage dependencies are only true for the model of the current architecture respectively target architecture.

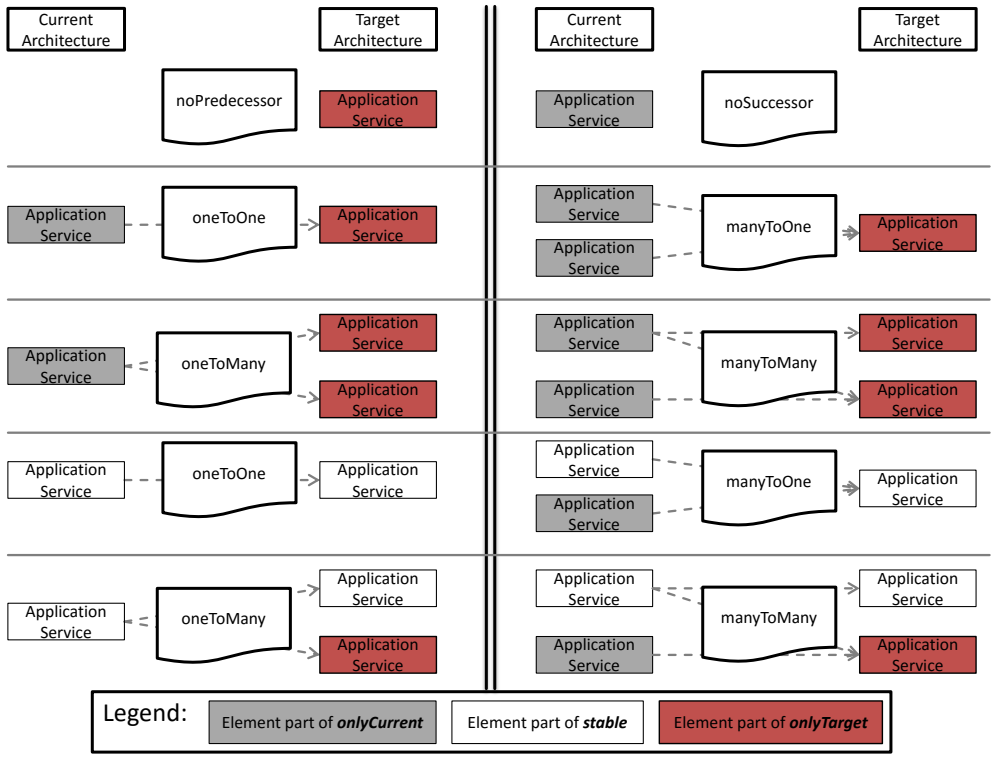

Figure 5.2: Transformation Patterns for Application Services

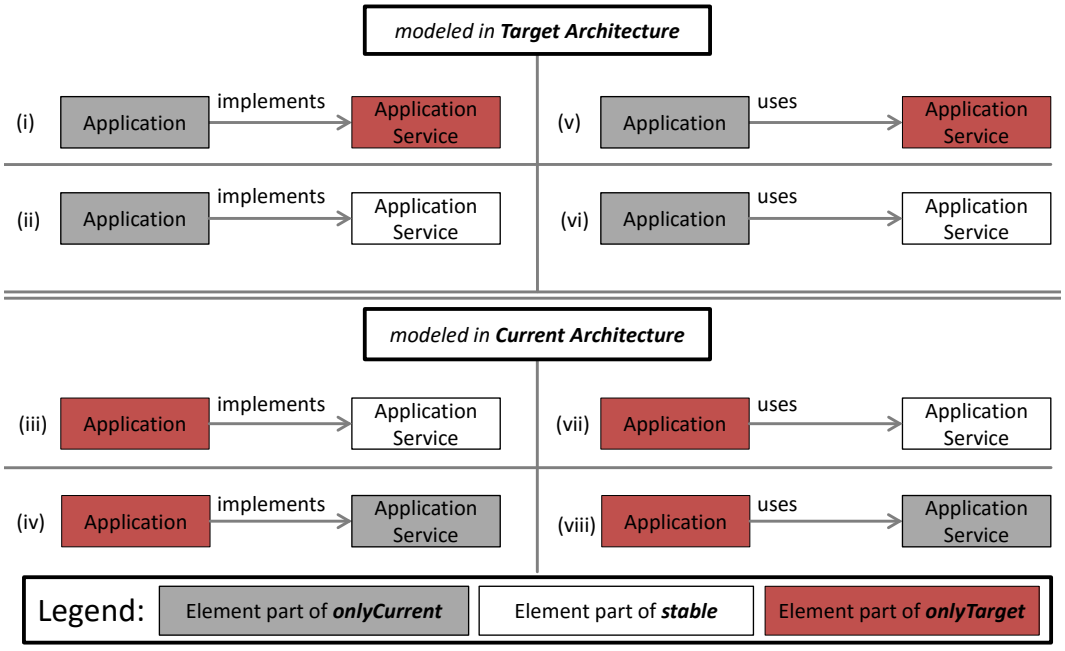

Figure 5.3: Ambiguous Situations in Transformation Path Generation

These ambiguous situations are avoided if the transformation actions from Chapter 4 are used to design EAG_{target}. Regarding the implementation relationships between applications and application services we decided to allow no change of the implementing application. Such a change is considered with a new or extended application service. We could provide transformation patterns for them if we allow for a change in the application that implements an application service. We do not provide transformation actions to change the implementing application of an application service, as we do not consider the application service as the same if the implementing application changes. Those changes are considered in our approach as new application services.

5.1.2.2 Transformation Actions for Transformation Path Generation

Given the transformation patterns for transformation path generation we need transformation actions that allow us to develop proposed applications and application services, and change the dependencies between them. Furthermore, we need to take care of all the applications and application services that are to be shutdown and the functional enhancements and decrements of the application services need to be considered (see Definitions 26 and 27). Therefore, we derive five different types of transformation actions that allow for a sequencing of changes. The types are 'develop application', 'develop application service', 'change dependencies', 'shutdown application service' and 'shutdown application' and they are logically ordered as shown in Figure 5.4. The logical ordering was already presented in Lautenbacher et al. (2013). However, the detailed mechanisms described in this paper were not fully explored and the formalisation was not fully elaborated.

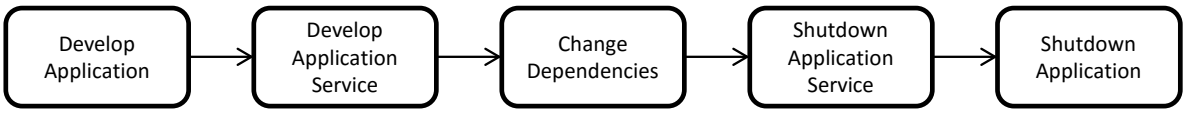

Figure 5.4: Logical Ordering of Transformation Actions; Introduced in Lautenbacher et al. (2013, Figure 3)

Possible sequences for the transformation actions are restricted by the logical ordering for the different types of transformation actions. At first, it is necessary to develop an application. Afterwards, we develop application services for the an application. We then change the usage dependencies between the applications and application services. After an application service is not used anymore we shut it down. As soon as every application service of an application is shutdown the application is shutdown, too. Please note, that the logical ordering does not postulate that every application needs to be developed via a transformation action to be able to develop the application services it implements. This is the case for applications that belong to *stable* and implement one or more application services that belong to $onlyTarget$. Then it is possible to select the 'develop application

service' transformation action right away. Furthermore, it is not necessary to shutdown every application and application service.

The types of transformation actions shown in Figure 5.4 provide a compact view on the transformation actions offered in transformation path generation. We need formalisms that take care of all underlying transformation patterns. Therefore, we introduce the set of transformation actions $P_{TP_{TA}}$ consisting of the following transformation actions for the different types. The LHS, RHS, and NAC of all transformation actions in $P_{TP_{TA}}$ are shown in section A.3 of Appendix A. In the following we introduce and describe them. For the two transformation actions $P_{DevelopAS}$ and $P_{ShutdownAS_{allSucc}}$ we show the LHS, RHS, and NAC also below.

Develop Application $P_{DevelopApp}$ allows one to change the lifecycle phase of an application from proposed to live. It is only applicable to applications that are in the lifecycle phase proposed. Furthermore, the transformation action adds 'stable' as a marker to the application service to prevent its shutdown later on during the transformation path generation.

Develop Application Service $P_{DevelopAS}$ has the same effect on an application service, but it is necessary that the application that implements the application service is in the lifecycle phase live. Figure 5.5 shows the LHS, RHS, and NAC of the transformation action $P_{DevelopAS}$.

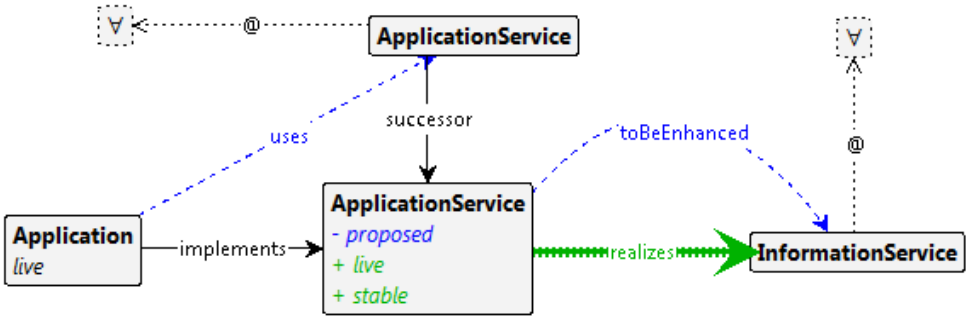

Figure 5.5: LHS, RHS, and NAC of Transformation Action $P_{DevelopAS}$

The condition that an application is live is either established through a transformation action that develops an application or the application belongs to *stable* and is already live. If the application that implements the application service used one of its predecessors this usage dependencies are removed. This is done because the functionality of the application service is part of the application as soon as the application service is developed and the usage dependency to the predecessors is no longer necessary. This holds for all usage dependencies of the application to the predecessors. Therefore, a ∀-Symbol is

connected to the predecessor application services that creates a nesting for one to many application services. If the application service, that is to be developed, has no predecessor $P_{DevelopAS}$ is applicable anyway. Furthermore, $P_{DevelopAS}$ takes the dependencies of the application service to the information services it realizes into account by adding 'realizes' edges. A stepwise development of an application services' functionality, i. e. an incremental development considering every 'realizes' edge to the information service separately, is not supported for application services that belong to *onlyCurrent*. This is why a ∀-Symbol is also connected to the information service.

Change Dependencies Using $P_{ChangeDepForSuccessor}$ allows to remove all dependencies of predecessor applications with the same successor application for a certain application service and create the dependency for the successor application. It is not allowed that the successor application service is implemented by the successor application. Furthermore, the application services' functionality needs to be already changed according to the target architecture, i. e. no enhancements or decrements for the application service are necessary. Changing dependencies for a certain application service to its successor is done with $P_{ChangeDepToSuccessorAS}$. It changes for all applications the usage dependencies from one application service to its successors. Furthermore, the change of the dependencies is limited to those applications that do not implement the successor application service and that no enhancements of the successor application service is necessary. This is ensured through the NAC of $P_{ChangeDepToSuccessorAS}$. The design rationale for this is that an application is not allowed to use an service it implements and that the functionality of an application service needs to be realized in order to provide the service properly for the applications that use it. Dependencies for application services that are not changeable through these two transformation actions are discussed in Subsubsection 5.1.2.3.

Considering necessary changes to the functionality of application services that belong to *stable* is done with the transformation actions $P_{EnhanceStableAS}$ and $P_{DecrementStableAS}$. The former adds functionality to an application service as specified in the target architecture, i. e. adding the realizes edges to the application service. The latter removes functionality. $P_{EnhanceStableAS}$ and $P_{DecrementStableAS}$ support a stepwise change of the application services' functionality, i.e. adding and removing the realizes edges from application services to information services, where it is necessary.

Shutdown Application Service For shutting down application services we provide three different transformation actions. Application services that do not have a successor and belong to *onlyCurrent* are retired with $P_{ShutdownAS_{noSucc}}$. It is necessary that the application service is not in use by an application anymore. This condition is also necessary to apply $P_{ShutdownAS_{succ}}$ and $P_{ShutdownAS_{allSucc}}$ for the cases where an application service has one or more successors. $P_{ShutdownAS_{allSucc}}$ retires as many application services as possible with the same successor application service.

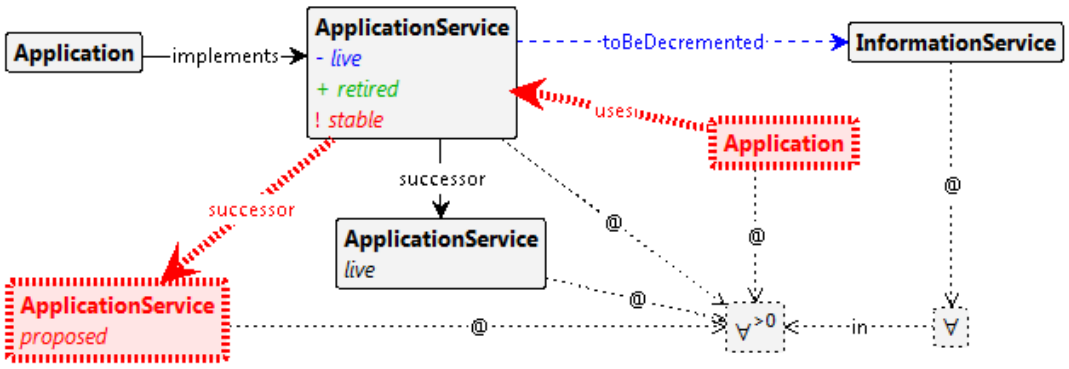

Figure 5.6: LHS, RHS, and NAC of Transformation Action $P_{ShutdownAS_{allSucc}}$

Figure 5.6 shows the LHS, RHS, and NAC of transformation action $P_{ShutdownAS_{allSucc}}$. The NAC specifies, besides that there is no using application for the application service, that no proposed successor is present and that the application service does not belong to *stable*. Applying $P_{ShutdownAS_{succ}}$ changes the lifecycle phase from live to retired. $P_{ShutdownAS_{succ}}$ has the same effect on an application service as $P_{ShutdownAS_{allSucc}}$, but just on one specific predecessor application service. Applying $P_{ShutdownAS_{allSucc}}$ has the same effect as repeatedly selecting $P_{ShutdownAS_{succ}}$, where the number of repetitions corresponds to the number of successor application services. All transformation actions that shutdown an application service remove also the functionality realized by an application service, as specified in the target architecture.

Shutdown Application The corresponding transformation actions to shutdown applications are $P_{ShutdownApp_{noSucc}}$, $P_{ShutdownApp_{succ}}$, and $P_{ShutdownApp_{allSucc}}$. Shutting down an application is only possible if none of its application services is used anymore. $P_{ShutdownApp_{allSucc}}$ retires as many applications as possible, in contrast to $P_{ShutdownApp_{succ}}$ which retires just one application. Changing the lifecycle phase from live to retired for applications that do not have a successor is done with $P_{ShutdownApp_{noSucc}}$. Applying $P_{ShutdownApp_{allSucc}}$ has the same effect as repeatedly selecting $P_{ShutdownApp_{succ}}$, where the number of repetitions corresponds to the number of successor applications.

5.1.2.3 Unique Dependencies

We need to add transformation actions that are bound to individuals in order to be able to create and delete unique dependencies. Figure 5.7 shows the different situations that lead to unique dependencies. These dependencies are known to be present, respectively not present, in the target architecture. However, they cannot be created or deleted through a change of the dependencies from the predecessor to the successor by using $P_{ChangeDepForSuccessor}$ or $P_{ChangeDepToSuccessorAS}$.

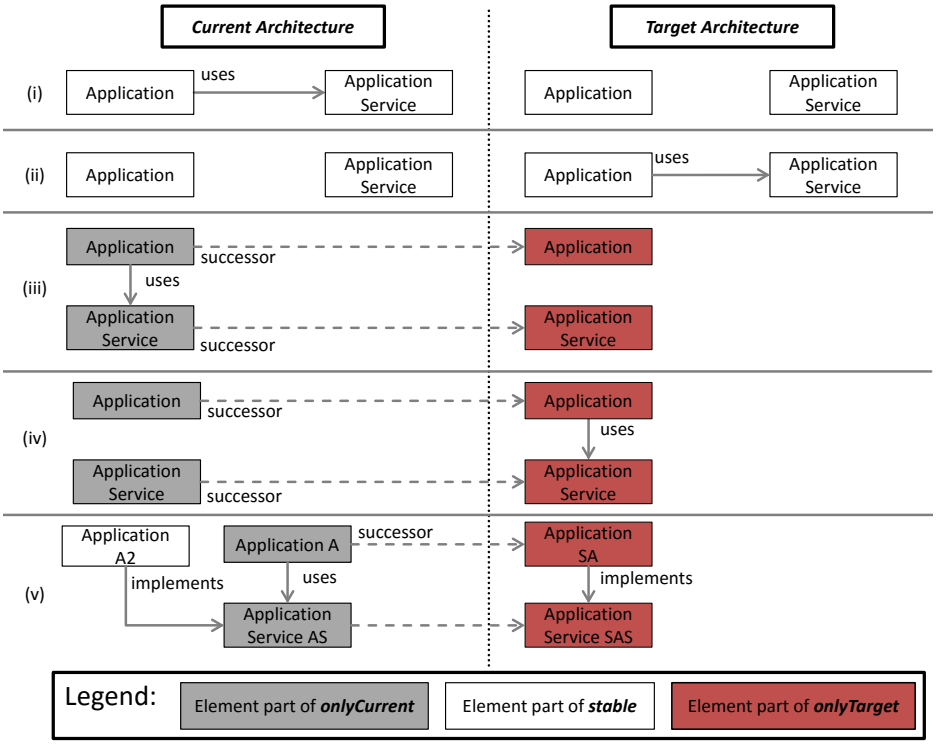

Figure 5.7: Unique Dependencies Between Current and Target Architecture

Unique dependencies exist when a dependency is present only at the current or target architecture and the dependent elements belong to stable (Figure 5.7, (i) and (ii)). Furthermore, if a predecessor application uses an application service in the current architecture and its successor application does not use the successor application service a unique dependency is present (Figure 5.7, (iii)). For the opposite case also a unique dependency is present where a predecessor application does not use a predecessor application service, but the successor application uses the successor application service (Figure 5.7, (iv)). The last case in which unique dependencies are present is when a predecessor application uses an application service which successor is implemented by a successor application of that predecessor application (Figure 5.7, (v)). To consider those changes we use transformation actions, bound to certain applications and application services, that form the set $P_{TP_{unique}}$.

As an alternative to the derivation of unique dependencies one could create transformation actions that allow to add and delete usage dependencies between any application and application service. The consequence would be that we need a heuristic whether the added or deleted edge is part of the target architecture or not. We decided to use the approach with the unique dependencies that can be regarded as a modeled heuristic that prohibits the addition and deletion of non target architecture conform usage dependencies.

A SPARQL Query, that queries the EAM ontology (see Appendix C), retrieves all usage dependencies that are considered as unique for the current architecture. The corresponding query is shown in the listing below.

Listing 5.1: Query for the Unique Uses Dependencies

```
SELECT * WHERE {
  ?nachfolgeAnwendung rdf:type lea:Anwendung.
  ?nachfolgeAService rdf:type lea:Anwendungsservice.
  ?nachfolgeAnwendung lea:AnwendungNutztAnwendungsservice ?nachfolgeAService.
  OPTIONAL {
    ?anwendung rdf:type lea:Anwendung.
    ?aservice rdf:type lea:Anwendungsservice.
    {?anwendung lea:AnwendungHatNachfolger ?nachfolgeAnwendung. }
    UNION
    {?anwendung lea:AnwendungIstNachfolgerVonSichSelbst ?nachfolgeAnwendung. }
    {?aservice lea:AnwendungsserviceHatNachfolger ?nachfolgeAService. }
    UNION
    {?aservice lea:AnwendungsserviceIstNachfolgerVonSichSelbst ?nachfolgeAService. }
    ?anwendung lea:AnwendungNutztAnwendungsservice ?aservice.
  }
  FILTER(!bound(?anwendung) || !bound(?aservice))
}
```

5.1.2.4 Initial State Transformation Path Generation

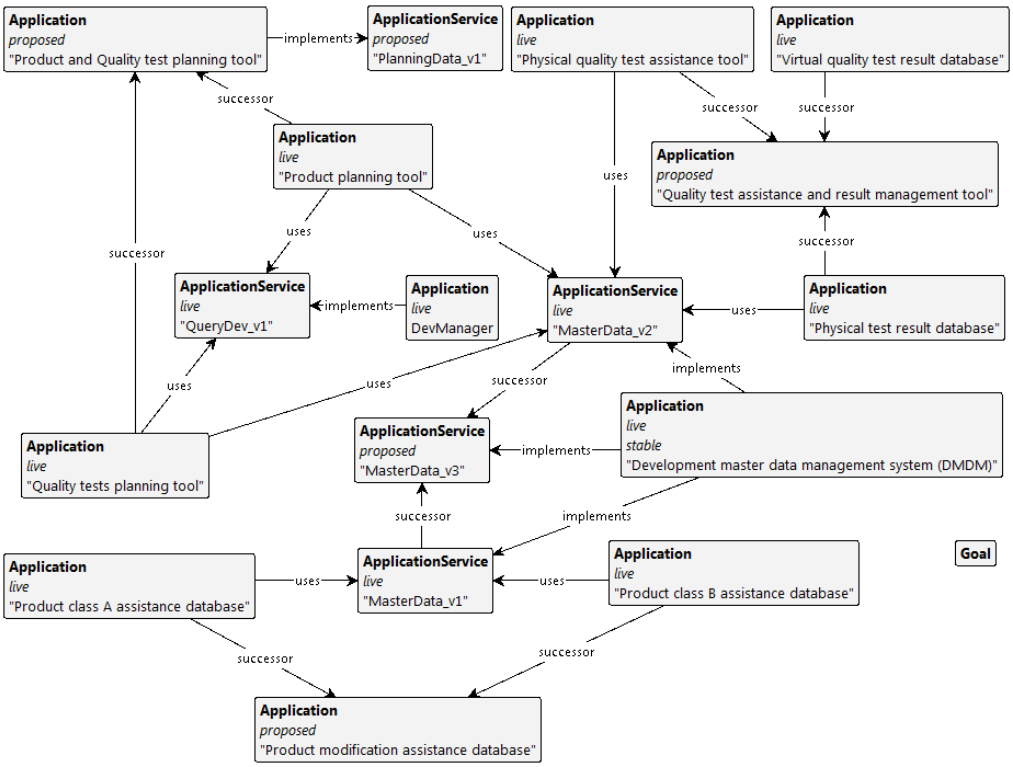

Figure 5.8: Initial State S_{TP} for Transformation Path Generation

We retrieve the initial state S_{TP} by querying the used EA tool. S_{TP} consists of all applications, application services and information services present in the current and target architecture and their respective lifecycle phases. The usage dependencies between applications and application services of the current enterprise architecture are also included in S_{TP}.

Figure 5.8 shows the initial state S_{TP} for the Development Master Data Use Case (see Section 7.4) used for transformation path generation. For further details on the context of this use case we refer to the description provided in Section 7.4. Please note that the goal node is already part of the initial state and will be marked as soon as the $P_{goalState_{TP}}$ is applied.

Furthermore, all 'realizes' edges between application services and information services are added to S_{TP}. 'Realizes' edges that belong to *onlyCurrent* are changed to 'to be decremented' edges and the realizes edges belonging to *onlyCurrent* are changed to 'to be enhanced'. Applications and application services that are present in both states are set to 'stable'. The information of changes in functionality of application services, and which applications and application services belong to *stable* is derivable for EA models, based on semantic web technologies, through a gap analysis (Diefenthaler and Bauer, 2013).

A SPARQL query, that queries the EAM ontology (see Appendix C), allows to retrieve the necessary information for the initial state. It is shown in the listing below.

Listing 5.2: Query for Initial State for Generating a Transformation Path

```
CONSTRUCT{
  \# All applications and application services in lifecycle phase "live"
  ?anwendung rdf:type lea:Anwendung.
  ?anwendung lea:hatLebenszyklusphase "live".
  ?aservice rdf:type lea:Anwendungsservice.
  ?aservice lea:hatLebenszyklusphase "live".
  \# All relationships between applications and applications services
  ?anwendung ?p ?aservice.

  \# All applications and application services in lifecycle phase live "propsed"
  ?nachfolgeAnwendung rdf:type lea:Anwendung.
  ?nachfolgeAnwendung lea:hatLebenszyklusphase "vorgeschlagen".
  ?nachfolgeAService rdf:type lea:Anwendungsservice.
  ?nachfolgeAService lea:hatLebenszyklusphase "vorgeschlagen".
  \# All successor relationships from the transformation model
  ?anwendung ?suc1 ?nachfolgeAnwendung.
  ?aservice ?suc2 ?nachfolgeAService.

  \# All implements relationships in the target architecture
  ?nachfolgeAnwendung lea:AnwendungImplementiertAnwendungsservice ?nachfolgeAService.

  \# All uses relationships in the target architecture
  ?nachfolgeAnwendung lea:AnwendungNutztAnwendungsservice ?nachfolgeAService.
}

WHERE {{
  \# All applications and application services including relationships between them, if any
  ?anwendung rdf:type lea:Anwendung.
```

```
    ?aservice rdf:type lea:Anwendungsservice.
30    OPTIONAL{
      ?anwendung ?p ?aservice.
32    }
    }
34  UNION{
    \# All successor relationships between applications
36  ?anwendung rdf:type lea:Anwendung.
    ?nachfolgeAnwendung rdf:type lea:Anwendung.
38  ?anwendung ?suc1 ?nachfolgeAnwendung.
    }
40  UNION{
    \# All successor relationships between application services
42  ?aservice rdf:type lea:Anwendungsservice.
    ?nachfolgeAService rdf:type lea:Anwendungsservice.
44  ?aservice ?suc2 ?nachfolgeAService.
    }
46  UNION{
      \# All implements relationships in the target architecture
48    ?nachfolgeAnwendung rdf:type lea:Anwendung.
      ?nachfolgeAService rdf:type lea:Anwendungsservice.
50    ?nachfolgeAnwendung lea:AnwendungImplementiertAnwendungsservice ?nachfolgeAService.
    }
52 }
```

5.1.2.5 Goal State for Transformation Path Generation

We create a typed attributed graph production $P_{goalState_{TP}}$ to determine the goal state, by querying the EA tool in use. It consists of all applications present in the current and target architecture and their lifecycle phases are set to the one in the target enterprise architecture. Furthermore, the usage dependencies between applications and application services of the target enterprise architecture are included. Applications and application services that are present in both states are set to stable. This goal state allows for a transformation path generation considering changes between the usage dependencies of applications and application services. In contrast to the typed attributed graph productions to detect goal states in Chapter 4, $P_{goalState_{TP}}$ is bound to certain individuals and the relationships between them. Nevertheless, $P_{goalState_{TP}}$ adds the reached attribute to the goal node to mark it.

Figure 5.9 shows an excerpt of the goal state of the Development Master Data Use Case (see Section 7.4) used for transformation path generation. The excerpt does not show the applications and application services to be retired. Furthermore, the information services that are realized by the application services are not shown.

The goal state is also retrieved via a SPARQL query. The only difference to the query for the initial state is that we have to query the applications and application services that are in the lifecycle phase live and retired in the target architecture. Furthermore, we are only interested in the usage dependencies that hold in the target architecture.

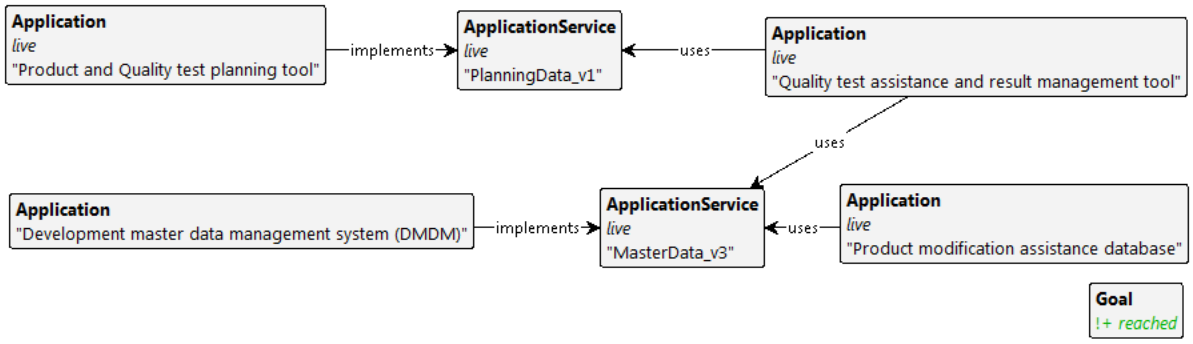

Figure 5.9: Excerpt of a Goal State for Transformation Path Generation

5.1.2.6 Reference Scenarios

In Subsection 2.1.4 we introduced reference scenarios as a best practice sequence of changes that should be followed in certain scenarios. Reference scenarios are described in detail in the Quasar Enterprise approach from Engels et al. (2008). If an enterprise architect wants to consider reference scenarios in transformation path generation, we have to extend the PDM and $DSIG_{PDM}$, and as a consequence the GTS_{TP} including transformation actions $P_{TP_{RS}}$ for the steps of the reference scenarios. Furthermore, S_{TP} must contain information that allows the execution of the additional transformation actions for reference scenarios.

GG_{RS} and its components are defined as follows:

- GG_{RS} is the typed graph grammar used for transformation path generation consisting of GTS_{RS} and S_{RS}
- GTS_{RS} as a typed attributed graph transformation system consisting of $DSIG_{RS}$, PDM_{RS} and P_{RS}
- $DSIG_{RS} = DSIG_{PDM} \cup DSIG_{Ext}$ with
  - $DSIG_{PDM}$ as defined in Subsection 3.5.4
  - $DSIG_{Ext}$ as introduced later in this paragraph and shown in Figure 5.12
- $PDM_{RS} = PDM \cup PDM_{Ext}$ with
  - PDM as defined in Subsection 3.5.3
  - PDM_{Ext} as introduced later in this paragraph and shown in Figure 5.11

- $P_{RS} = P_{TP} \cup P_{TP_{RS}}$ with
  - P_{TP} as introduced in subsubsection 5.1.2.2 except $P_{DevelopAS}$
  - $P_{RS_{DevelopAS}}$ is the same as $P_{DevelopAS}$, but with an additional NAC that prohibits its execution when a reference scenario was selected (see Figure A.32 in section A.4)
  - $P_{TP_{RS}}$ as the transformation actions for the steps of the two reference scenarios we support
- $S_{RS} = S_{TP} \cup S_{Ext}$ as the initial state of GG_{RS} with
  - S_{TP} as introduced in subsubsection 5.1.2.4
  - S_{Ext} as the additional information necessary in S_{RS} to allow the application of $P_{TP_{RS}}$

$P_{goalState_{TP}}$ is not influenced by the introduction of the reference scenarios, as the reference scenarios have a predetermined result inherent to the target enterprise architecture. Reference scenarios are preferable detours considerable in a typed graph grammar used for transformation path generation. Please note that we cannot provide any changes to the transformation actions present in $PD_{selected}$ automatically. We assume the factors, that are relevant for reference scenario selection, to be considered by the enterprise architect in the selection process of the transformation actions.

We support the following two reference scenarios through transformation actions, as introduced in Subsection 2.1.4:

- Develop supporting application services before core business services
  - step two: temporarily provide functionality to the newly created application; formalised as $P_{RS_{SS2}}$
  - step three: reverse dependency between newly created application and other application; formalized as $P_{RS_{SS3}}$
- Develop inventory applications first
  - step two: establish a temporary data integration between the inventory component and the application that implements the inventory functionality; formalized as $P_{RS_{IC2}}$
  - step three: replace the temporary data application service through a logic

application service; formalized as $P_{RS_{IC3}}$

– step four: remove the temporary data dependency and reverse it by establishing a dependency to the logic application service; formalized as $P_{RS_{IC4}}$

A reference scenario is a sequence of transformation actions that is considered as a best practice way of changing certain differences between a current and target architecture. Every step of each reference scenario is formalized with its own transformation action. They form the set of reference scenario transformation actions $P_{TP_{RS}}$. Please note that the first step of both reference scenarios is already established through a transformation action that develops an application ($P_{DevelopApp}$). The LHS, RHS, and NAC of the transformation actions of all reference scenarios are shown in the Figures A.27 to A.31. In the following we describe the LHS, RHS, and NAC of $P_{RS_{IC2}}$ in detail as shown in Figure 5.10.

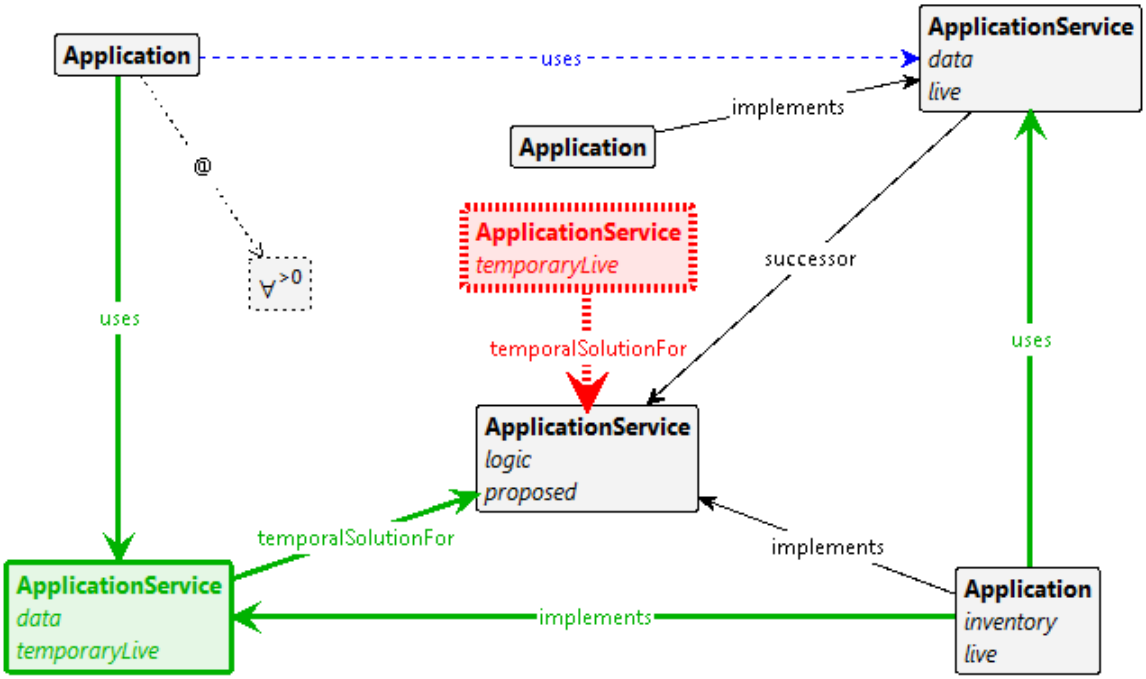

Figure 5.10: LHS, RHS, and NAC of Transformation Action $P_{RS_{IC2}}$

$P_{RS_{IC2}}$ is the second step for the inventory components first reference scenario. The transformation action $P_{RS_{IC2}}$ is applicable if an application service, of the type *logic* is implemented by an application of category type *inventory*. Furthermore, an application service of type *data* is the predecessor of the *logic* application service. The NAC of $P_{RS_{IC2}}$ prohibits that more than one reference scenario for the *logic* application service can be selected after it has been selected once. All applications that use the *data* application service are redirected to an application service that is created through $P_{RS_{IC2}}$. The new application service is set to temporarily live and is implemented by the *inventory*

application. Furthermore, an edge between the new application service and the *logic* application service is created to indicate that a temporal solution is created. This edge prohibits the recursive selection of the transformation as it is part of its NAC. Additionally, the edge prohibits the selection of $P_{RS_{DevelopAS}}$, as it is part of the NAC of $P_{RS_{DevelopAS}}$ (see Figure A.32).

Figure 5.11 shows PDM_{Ext} with the necessary extensions to the applications and application services. Temporary flags are used to mark applications and application services. Application services that are temporary created by $P_{RS_{SS2}}$ are marked with 'temporary live' to indicate that the application service is not to be shutdown by a transformation action of the type 'shutdown application service'. Furthermore, the application service modeled in the target architecture is marked with 'temporary not proposed' to prevent the selection of 'develop application service'. The types for application services allow to differentiate between 'logic' and 'data' application services, that are relevant for the 'inventory components first' reference scenarios.

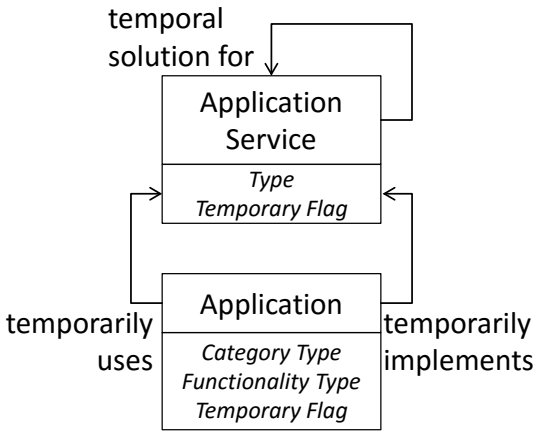

Figure 5.11: Extended Type Graph PDM_{Ext}

In the reference scenario 'supporting services before core business services' the applications are marked with the functionality types 'core and support functions', 'support functions' and 'core functions'. The application with the core functionality has the temporary flag 'support function to be removed' to indicate that the support functions need to be removed after the application with the support functionality was developed. Both applications are successors of the application with the mixed functionality, i.e. the application with the functionality type 'core and support functions'. A category type allows to mark applications as 'inventory' applications which is necessary for the 'inventory components first' reference scenarios. The necessary extensions are considered in $DSIG_{Ext}$. Figure 5.12 shows the final data signature $DSIG_{Ext}$ that forms together with $DSIG_{PDM}$ the final data signature $DSIG_{RS}$.

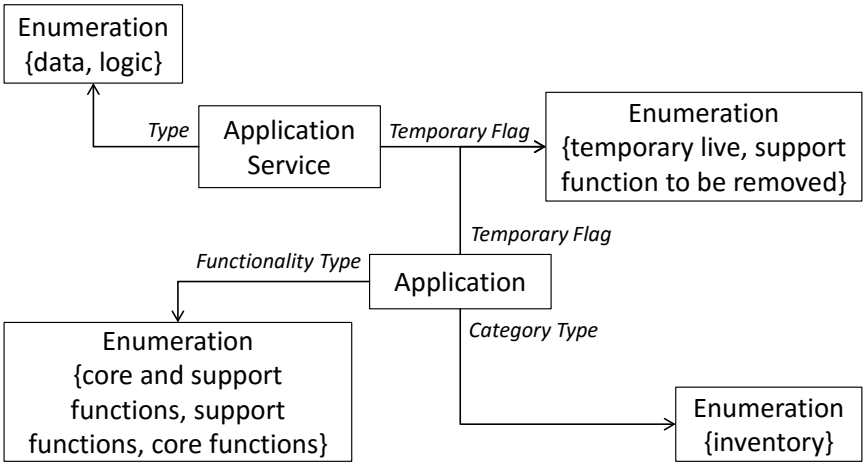

Figure 5.12: Extended Final Data Signature $DSIG_{Ext}$

5.2 Interactive Process for Transformation Path Generation

In this section we describe the process the enterprise architect allows to select transformation actions in transformation path generation. We only differentiate between the approach using reference scenarios and the approach without them where necessary. All parts that explicitly refer to the extended graph grammar GG_{RS} only hold for the extended approach using reference scenarios. Cases where the reference scenario approach is not explicitly mentioned also hold for the reference scenario approach ($GG_{TP} \subseteq GG_{RS}$).

5.2.1 Compute Applicable Transformation Actions

Given the initial state S_{TP} we compute all transformation actions applicable to it. The list of all applicable transformation actions is shown to the enterprise architect. All the transformation actions shown to her are potentially executable in parallel except the transformation actions that are alternatives to each other.

The list of transformation actions is updated after each selection and allows to determine if new transformation actions become available or are not available any longer.

5.2.2 Select an Applicable Transformation

The enterprise architect selects one of the transformation actions and assigns it to a step.

Definition 30. ***Step****: A step is a logical containment of transformation actions executable in parallel. The duration of a step is given through the transformation action, that is attached to the step, with the longest duration. Every transformation action is attached to exactly one step. A step has at least one transformation action attached to it.*

Furthermore, she can rearrange the stages a step belongs to.

Definition 31. ***Stage****: A stage is a logical containment of steps. Its duration is determined through the duration of the steps attached to it. A stage has at least one step and every step is attached to exactly one stage. After all transformation actions attached to a stage, via steps, have been conducted the changes are considered as finished and result in an intermediate (planned) architecture.*

Selecting the first transformation action automatically creates a step for the transformation action and a stage for the step.

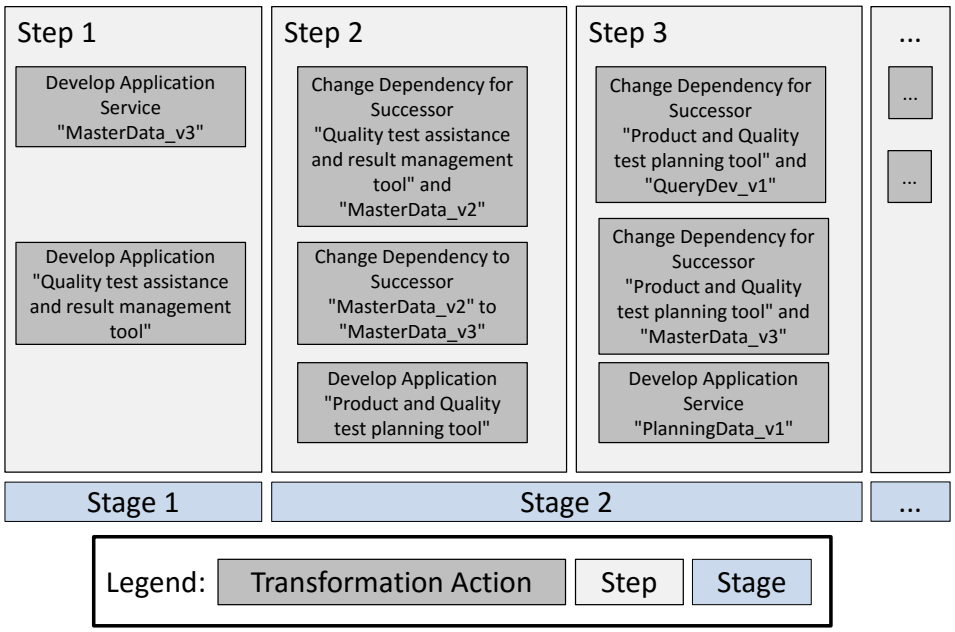

Figure 5.13: Assignment of Transformation Actions to Steps and Stages

Figure 5.13 shows the assignment of transformation actions to steps and stages for a certain S_{TP}, as shown in Figure 5.8.

Every selected transformation action gets a step number as an attribute. All transformation actions that where applicable in the initial state get the step number one. Every new applicable transformation action gets the step number increased by one. This allows the enterprise architect to assign transformation actions to steps.

We cannot provide a ranking for the different selectable sequences and assignment to steps and stages, as it is not possible to determine automatically the value of a transformation action performed at an earlier or later step.

If $P_{RS_{IC2}}$ or $P_{RS_{SS2}}$, for the second step of a reference scenario, is applicable it can be selected by the enterprise architect and considered in the transformation path generation.

As soon as $P_{goalState_{TP}}$ is applicable it is automatically selected and the enterprise architect is notified that the transformation path generation is successfully finished.

5.3 Output of the Interactive Transformation Path Generation

As a result we created the following information:

- every selected transformation action (instances of P_{RS}) is assigned to a step
- created intermediate (planned) architectures via stages (potentially all graphs G_i created through the application of P_{RS} on S_{TP})
- applied transformation actions and reference scenarios in the transformation path from the current to the target architecture

This information allows the enterprise architect to create a proposal for a roadmap that is to be decided upon and can be integrated with ongoing and already budgeted projects.

5.4 Summary of Interactive Transformation Path Generation

In this chapter we presented our approach for interactive transformation path generation. Given a ranking of transformation actions, a transformation model and the corresponding current and target architecture we showed how it is possible to create transformation paths interactively. Besides, we introduced the necessary formalisms for the graph grammar that allows to support the selection process and showed that it is possible to consider reference scenarios, as preferred detours, in the transformation paths.

6 Related Work

In this chapter we present and discuss related work of the thesis. The related work is divided into approaches that have different foci regarding their support in transformation path generation. Besides, purely model focused approaches we present tool supported approaches and approaches to optimize the decisions to made upon changes involved in a transformation path. Furthermore, we present related work that uses patterns in the context of EAM. For every related work we will explain how the approach works and what the differences and potentially drawbacks in comparison to our approach are.

6.1 Modeling Approaches for Planning Purposes in EAM

The modeling approaches to transformation path generation presented in the following all are based on a manual creation of the models by an enterprise architect. This is due to the origin of these approaches that focus on a modeling freedom for the creator of the models. The goal of these approaches is to support the enterprise architect in the modeling process, by allowing to add information relevant for transformation paths or restricting the possible models that can be created. The latter is realized by means of restrictions and extensions of the metamodel in use. However, the enterprise architect is not supported by automatic means to create models, for example a target architecture.

6.1.1 Action-Threat-Opportunity Trees

The Action-Threat-Opportunities (ATO) Trees approach allows to assess risks and opportunities in a project plan for changing an enterprise architecture (Sousa et al., 2013). Figure 6.1 shows an abstracted ATO tree with its different node types and hierarchy levels.

A goal, for example profit, is supported by several architecture principles which in turn are realized by actions. Architecture principles allow to restrict the design choices (Greefhorst and Proper, 2011) and are also depicted as goal nodes in Figure 6.1.

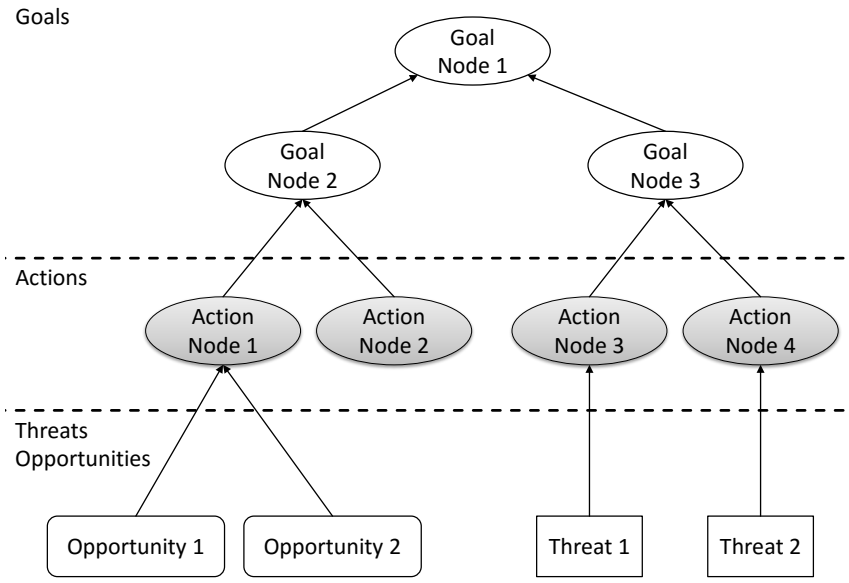

Figure 6.1: Schematic Action-Threat-Opportunity Tree; Abstracted from Sousa et al. (2013, Figure 5)

Actions may have risks or opportunities attached to them. An example for an action is the development of a new application. Different types of nodes in the tree are the hierarchical levels of a planning tree. Nodes are either exclusively disjunctive or conjunctive alternatives that constitute to the upper hierarchy element. Considering risks and opportunities, as the elements at the lowest hierarchy level, allows to calculate the benefits of actions and therefore to determine the most valuable set of actions.

After modeling the goals at the top the enterprise architect determines possible actions that support the achievement of the goals. The actions are also modeled manually in the tree and are refined through risks and opportunities that might result from executing the action. Then the enterprise architect determines the risk value for threats by choosing between 'low', 'medium', and 'high'. Furthermore, she determines the value of benefit for every opportunity. As a consequence every action of the ATO tree has now a value attached to it. This allows to aggregate the values up in the tree until a value is determined for the tool of the ATO tree. The aggregation considers whether the hierarchical lower nodes are exclusively conjunctive or not. By extending the ATO tree with countermeasure nodes the enterprise architect can attach countermeasure to threat nodes, as an a priori response to soften the consequences of the threat.

Differences to our approach. ATO trees provide the flexibility to model any actions the enterprise architect wants to consider and determine the best set of actions within the tree. Furthermore, exclusive alternatives of actions can be considered and countermeasures can be defined. Besides, the aggregation of the values and the determination of the best set of actions the approach is completely based on manual modeling.

The enterprise architect always has to model the different (individual) actions from scratch for the given goals. A reuse like for the transformation actions in different situations is not possible. Furthermore, the derivation of the resulting model after applying certain actions is not supported in the ATO approach. Resource constraints that may hinder the execution of actions or a combination of actions is also not supported. Additionally, our approach provides the possibility to assign preferences to different dimensions involved in the decision making. The ATO tree approach does not support such explicit preferences.

6.1.2 Modeling Transformation Paths using a Transformation Model

This approach focuses on the explicit modeling of successor relationships within a transformation model to allow for using network planning techniques and determining critical paths in a transformation path. Planning transformation paths in this approach uses a gap analysis as a starting point by identifying the differences between two states considering changed elements (Gleichauf, 2011, Figure 4). However, it is necessary to decide a priori on a target architecture to be realized. The approach distinguishes between a macro and micro level of enterprise architecture states, e.g. a current and target architecture, where on the micro level detailed information about successor relationships of the elements is available (Aier and Gleichauf, 2010b). Furthermore, information about changed relationships is part of the transformation model.

The approach suggests to compare the graphs of the different architecture states to gain information about necessary changes (Aier and Gleichauf, 2010a). Afterwards, it is possible to derive six different types of successor bundles between the elements of the different architectures and this information is stored in a transformation model. The types are one to none, none to one, one to one, many to one, one to many, and many to many. It is possible that each of the types is instantiated in the transformation model one or more times, or is even not instantiated. The case where an element A is a successor of an element B, where A and B are part of an enterprise architecture at a former point in time, is not considered. For every enterprise architecture at different points in time a transformation model exists (Aier and Gleichauf, 2010b). Changing dependencies between elements is not considered as part of the changes, even though they are part of the transformation model. With a transformation model it is possible to identify necessary changes, to sequence them and to apply the critical path method afterwards.

Differences to our approach. The presented approach only allows for a consideration of the same elements within the successor type one to one. Therefore, we extended the transformation model to allow the same element being part of other types of relationships

(see Figure 3.2), as we were faced with situations in the use cases where such relationships were present. Our approach does not explicitly provide the means to conduct a critical path analysis, however the critical path is given implicitly, as the transformation actions with the longest durations form the critical path. Besides, considering different resource modes our approach provides the advantage of automatically deriving the transformation model by selecting transformation actions. In contrast our approach does not provide means to refine the transformation actions further. Nevertheless, our transformation actions could be extended to allow for a selection of, for example, transformation actions that allow for considering the testing phase of an application. An insight from our results is that the decision upon a target architecture is coupled to the decision upon the transformation model. We consider this interrelation within the decision support for target architecture selection and allow for a resequencing of transformation actions in the path by automated means.

6.1.3 ArchiMate

Another standard from the Open Group in the domain of enterprise architectures is the modeling language ArchiMate (The Open Group, 2012). Even though the ArchiMate concepts are different from the ones defined in the TOGAF content framework a combination in practice is possible. ArchiMate provides in its current version for each concept a visual notation, that is intended to help the enterprise architect to understand the visualizations of the enterprise architecture faster and better.

ArchiMate (The Open Group, 2012, Chapter 11) introduces an Implementation and Migration Extension including a gap element. Figure 6.2 shows the elements and relationships involved in the extension.

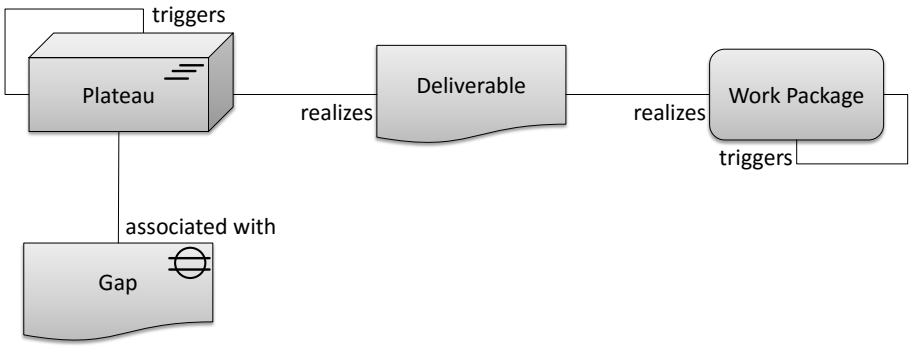

Figure 6.2: Elements and Relationships of the ArchiMate Implementation and Migration Extension; Adapted from The Open Group (2012, Figure 73)

The extension for the ArchiMate metamodel allows to explicitly model plateaus, gaps, deliverables and work packages. The latter are defined as “a series of actions designed to

accomplish a unique goal within a specified time" (The Open Group, 2012, p. 145) and can be projects that realize changes in the enterprise. Deliverable are the results of work packages, but can also be used for creating work breakdown structures used in project planning. If a deliverable is realized successfully, through a work package, the state of the enterprise architecture changes. ArchiMate calls these states 'Plateaus'. They differentiate between current (baseline) architecture, target architecture, and transition architectures. The current and target architetcure are in accordance with our understanding of these states as described in Section 3.1. Transition architectures are the intermediate enterprise architecture states and may have alternatives to each other. The gap element is defined as "an outcome of a gap analysis between two plateaus" (The Open Group, 2012, p. 148). ArchiMate does not detail their conception of the gap analysis and tasks to be laid out.

Even though ArchiMate does not provide a methodology for the creation of the models the existing definitions give hints regarding the order in which the results are created. To be able to perform a gap analysis the enterprise architect needs to model a target architecture, respectively a transition architecture. We assume that the creation of the target architecture is influenced by the deliverables that are deduced from requirements, that are not depicted in Figure 6.2. This means that the deliverables are known before the plateaus are modeled. Based on the deliverables the work packages are then defined.

Differences to our approach. ArchiMate is a modeling language that introduces concepts for enterprise architecture metamodel elements. To the best of our knowledge no tool support with automated mechanisms, to derive for example plaeteaus, is available. The interconnection between work packages and plateaus is not explicitly representable like in our approach between transformation actions and the resulting states. ArchiMate provides neither a possibility to integrate a ranking regarding the flow from one plateau to another, nor does it consider different resource employments. Transition architectures seem not to be derivable automatically given a current and target architecture. Our gap analysis is metamodel independent which means that it is possible to automatically derive all types of gaps regardless of the metamodel in use and for all types of gaps. ArchiMate in contrast considers only gaps for elements (The Open Group, 2012, Figure 78).

6.1.4 Dimensions of EA Transformations and Their Consistency Constraints

Buckl et al. (2011) point out the importance of modeling enterprise architecture transformation aspects and derive different dimensions of modeling those transformations. The dimensions are created through the aspects which allow a refinement of the pe-

riod of validity of entities. Aspects are differentiated into the driving activity (A), the replacement relationship (R), and the lifecycle information (L). Figure 6.3 shows the dimensions of enterprise architecture transformations.

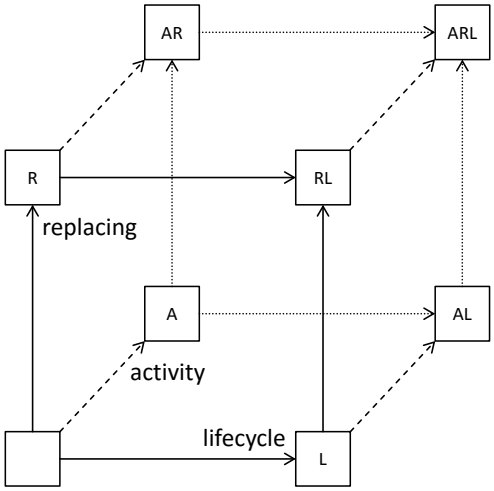

Figure 6.3: Aspects of EA Transformation Modeling; From Buckl et al. (2011, Figure 1)

A driving activity is the entity or a group of entities that carry out changes to other entities, for example projects or programs that change applications. One or more entities are replaced by one or more entities which is modeled through the replacement relationship. Replacements take into account the successor relationships of the transformation model as presented in Subsection 6.1.2. Lifecycle information is used for non-rigid entities to denote that they are only valid for a specific period in time which is important from an ontological point of view. A non-rigid concept is one that may not hold over the entire lifespan of an entity. In contrast a rigid concept is for example a node of the type graph of the PDM presented in Figure 3.7. Once an entity is an application it will always be an application. However, some of the attributes present in the PDM can be considered as non-rigid, as for example the schema entries of an information object might change.

Another, important ontological aspect taken into consideration is the identity of entities that may be given through a concept. Being not relevant for an entities identity is accounted for by differentiating between dispersive and non-dispersive concepts. Furthermore, Buckl et al. (2011) introduce synchronic and diachronic relationships. Relationships that hold independently of the time of validity for an element are diachronic. Synchronic relationships in contrast are only valid where the time of validity intersects for elements of the enterprise architecture. All of these ontological aspects need to be considered in the creation of models for transformation. However, the approach suggests to add constraints to the metamodel in use to formalize the ontological aspects. How these consistency constraints are formalized is shown based on the Object Constraint Language (International Organization for Standardization (ISO), 2012). Therefore, the approach is capable of specifying that, for example, the validity of an entity is defined

through the start and end dates of the activities that introduce and retire it. Regarding modeling replacements via successor relationships the constraints allow no cycles in the relationships and that requires that the validity of an entity, that replaces another entity, is not longer than the validity of the replaced element.

Differences to our approach. The presented approach provides a formalization of well established ontological concepts. However, only a constraint perspective regarding valid models is considered that assists in finding incorrect models. This constrains the number of possible valid models, but cannot assist an enterprise architect in finding valuable models, respectively valuable transformation actions like in our approach. Furthermore, the authors of the approach state that "a user willing to establish EA planning in a practice-relevant way on a rigorous basis, is faced with a situation that he or she has to decide which of the aspects is most important, being able only to select one very aspect for implementation." (Buckl et al., 2011, p. 156). We assume this situation to be resolved by the proposed metamodel extensions and consistency constraints proposed by the approach. However, we could not identify the consideration of changing relationships that have to be considered, as they are also time dependent and might result in higher or lower efforts in the transformation path. Lastly, our approach provides the means to distinguish the points in time even though the state of the architecture is the same, as we allow for a consideration of different resource modes for the transformation actions.

6.2 Tool Support for Planning Purposes in EAM

In this section tool support for planning is presented. Every tool is coupled to an enterprise architecture methodology and to a certain degree to a metamodel. This is a consequence of the fact that model and method can be considered to be the 'two sides of the same coin' (Winter et al., 2009), that are instances of the former meta concepts. We only consider tools that are affiliated with scientific publications, as the information on them is publicly available. Nevertheless, the Iteraplan and Enterprise Architecture Management System presented in this section are tools that are used by enterprise architects from different industry sectors.

6.2.1 OFFIS Prototype

The Institute for Information Technology of the University of Oldenburg (OFFIS) provides a tool supported approach for performing a gap analysis on a current and ideal landscape (Postina et al., 2009). Tool and approach are tightly coupled to the Quasar Enterprise approach, which can be used to develop service-oriented application landscapes (Engels et al., 2008).

We consider an application landscape, to be a model that is part of the enterprise architecture. The main functionality of the tool is to model a current architecture and an ideal landscape that are then compared with a gap analysis and allow for the derivation of an action list. We will call the ideal landscape in the following ideal architecture. It serves as a visionary future state that may never be realized, however it is very detailed, as we will explore in the following.

The current architecture in the OFFIS tool consists of current components, current services, current operations of the services and business objects. The ideal architecture is modeled with ideal components, ideal services, ideal operations and domains. Based on these two models the tool is capable to generate a list of actions that would, if all were applied, result in the ideal architecture. Within the tool the suggested procedure for selecting actions is to allow an architect to select actions from the list, but not all, that result in a target architecture.

Gringel and Postina state that the gap analysis needs a "detailed level of description when it comes to modeling both landscapes" (Gringel and Postina, 2010, p. 283) and as a result the "data necessary to perform gap analysis on the entire application landscape on a detailed level considering operations is overwhelming" (Gringel and Postina, 2010, p. 291). How the different actions interfere with each other cannot be considered and actions can only be provided if an ideal landscape with all details has been modeled.

The OFFIS prototype is capable of providing metrics for quantitative analysis of the models (Gringel and Postina, 2010). For example the purity of domains can be computed and allows to determine the applications that provide operations that are used in more that one domain. A weighting of metrics is possible and can be reused. Given the weight and models of current and ideal architecture "the tool calculates the structural deviation" (Gringel and Postina, 2010, p. 289). The metric of each action is bound to its structural deviation and the preferences are only determinable for the overall metrics.

Differences to our approach. Our approach allows a decision support for target architecture selection without having an ideal architecture on a detailed level. Furthermore, our approach considers the interdependencies of transformation actions. Successor relationships are neither considered in the approach nor the tool. We allow for their creation in the selection of a target architecture. The gap analysis in the OFFIS tool is not metamodel independent, in contrast to our set theoretic approach to comparison. Within the OFFIS tool it is only possible to conduct a gap analysis based on the Quasar Enterprise metamodel. Through the feedback loop viewpoint introduced in our approach we are even able to provide insights on transformation path deviations and successful transformations, for measuring continuous architecture progress regardless of changing target architectures (see Section 3.3). Furthermore, our approach provides transformation actions for sequencing changes within the transformation path and needs no ideal architecture on the same level of detail as the current architecture, which makes it less approach-dependent. Ranking transformation actions in our approach considers different

instance dependent factors, i. e. for information objects, in contrast to the OFFIS tool where a order for the different supported metrics is realized but not for each attached actions.

6.2.2 Serviceoriented EAM-Tool

The Serviceoriented EAM-Tool (SEAM-Tool) supports a method to manage service and process oriented enterprise architectures by using a case based reasoning approach and a case repository to provide information for the decision support of an enterprise architect (Postina, 2012). Using the tool allows for establishing a 'Serviceoriented Enterprise Architecture Management'.

The case based reasoning allows to reuse viewpoints that are considered useful by enterprise architects, as they were used in past planning processes for decision making. Given the constellation of involved entities of the past the case based reasoning allows to generate the views for a new planning problem. Figure 6.4 shows the concepts involved in the SEAM-Tool to create the views for the enterprise architect in new planning iterations.

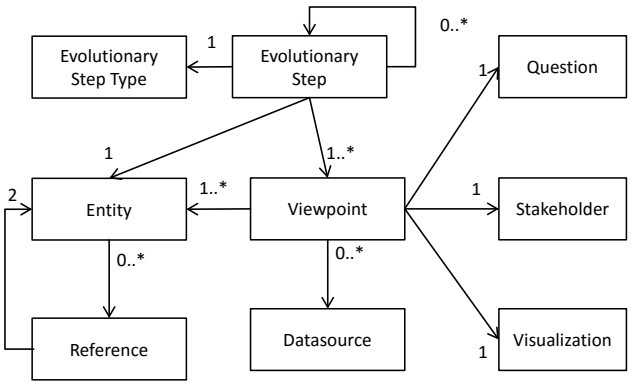

Figure 6.4: Different Concepts in the SEAM-Tool; From Postina (2012, Figure 4.2)

The target architecture is reached from the current architecture through several evolutionary steps. Each evolutionary step is a planned change. A case is defined by the type of an evolutionary step, the entities and their types involved, for example an organizational unit headquarters, and the viewpoints attached to the evolutionary step. An evolutionary step type can be either 'create', 'update' or 'delete' regarding whether the evolutionary step introduces new entities, or deletes entities, or changes existing entities. The cases help to provide views, named visualizations in Figure 6.4, for future evolutionary steps with the same combination of entity types and evolutionary step type. An evolutionary step can consist of several evolutionary steps. The enterprise architect

creates transformation paths through evolution scenarios. Each scenario consists of one or more evolutionary steps.

The case based reasoning consists of four activities that help to support the enterprise architect in defining evolutionary steps. At first a selection of similar cases is created and selected. Then the existing knowledge of the selected cases is reused by modifying it to the evolutionary step that is to be created. Afterwards, the reused and adapted evolutionary steps are revised by the enterprise architect. At last the case is retained if the created case is satisfactory for the enterprise architect and a reuse in the future is possible. The question element attached to the viewpoint allows the reuse of a case and the stakeholder element is used to group the cases accordingly to the roles of the users. Viewpoints are connected with datasources that allow for a creation of the views in forms of visualizations.

Differences to our approach. The SEAM-Tool provides a good mechanism for reusing viewpoints that showed their usefulness in the past. However, when it comes to the decision support neither the derivation of transformation actions from current models, nor the creation of possible target architectures is supported. Furthermore, even though it is possible to sequence evolutionary steps, it is impossible to show complex temporal interdependencies between them. Lastly, we consider the restriction of evolutionary step types to 'create', 'update' and 'delete' to be to restrictive, as evolutionary steps can be composed of others. Transformation actions allow even for the expression of negative conditions (as part of its NAC) to be applicable, that is not possible for the evolutionary steps. However, we consider this information to be important in situations, as it might change the estimation of a situation present in a model. Nevertheless, the idea of abstracting from concrete models to the types of entities involved is similar to the abstraction mechanism of the transformation patterns. Though, we allow for assigning a meaning to the transformation patterns and do not aggregate them within one viewpoint, but allow for the assignment of several transformation actions to automatically support the creation of target models. The latter is explicitly not supported by the SEAM-Tool where the decision support is limited to a created visualization of the current architecture, but not regarding the target architectures (Postina, 2012).

6.2.3 SEAMCAD

The Systemic Enterprise Architecture and Methodology (Wegmann, 2003) combines the living systems theory of Miller (1978) and the Reference Model for Open Distributed Processing (RM-ODP) (Raymond and Armstrong, 1995) to allow for an effective alignment of the IT and business within an enterprise. SEAMCAD is the corresponding tool for the Systemic Enterprise Architecture and Methodology approach that allows to model enterprise architectures.

Systemic Enterprise Architecture and Methodology introduces an ontology to define the concepts and relationships to be considered in the enterprise architecture. Five basic modeling characteristics build the foundations of the ontology: object, action, state, location time, and location in space. The action is not used to model changes to the architecture, instead it allows to model the exchange of objects or changes to their states. Two specification characteristics allow for a distinction between type and instance.

Systemic Enterprise Architecture and Methodology also introduces a method to allow for the adaptation of the models created during the execution of projects, as a continuous evolution of the models in iterations is presumed. Enterprise architects test the models and validate them with domain experts. The iterative approach considers three different activities: multi-level modeling, multi-level design, and multi-level deployment. Modeling is used to create and update models, whereas the design identifies gaps and tries to resolve them. Deployment is concerned with the implementation of the changes that remove the gaps. The implementation is not limited to IT development activities and also considers necessary changes in the business. SEAMCAD implements the ontology and method.

Dam et al. (2010) present a tool supported approach for change propagation to allow for an evolution of the enterprise architecture. It is an extension to SEAMCAD. They provide formal repair operations based on Alloy[1] models to restore the consistency in the models after an enterprise architect has deleted elements or connections.

Alloy is a language which expressiveness is in first order logic and Alloy models are a collection of constraints to express a metamodel. A ripple effect of deletions in a model may cause such inconsistencies, i.e. that relationships dangle due to the deletion of elements within the model. The inconsistencies can be reasoned in the models created by an enterprise architect based on the Alloy models. This is possible as the models are restricted by first order logic formulae.

Repairing a model, after an inconsistency has been reasoned, is done with the repair operations, that remove the inconsistency. Available repair operations are computed and form a repair plan. A repair plan has an abstract syntax and is stored in a library, that allows to create the instances of repair plans. All possible repair plans are selectable by an enterprise architect to restore the consistency in the model, that have been modeled by her before. Therefore, the different repair plans allow her to decide upon different repair actions, or different combinations thereof, using her implicit knowledge.

A further extension of the tool and approach is the consideration of a cost based belief-desire-intention agent framework for repair plan selection (Dam and Winikoff, 2008). The repair operations, in Dam and Winikoff (2008) called actions, have costs that are defined through the type of repair operation performed.

[1] http://alloy.mit.edu/alloy/

Calculating the costs for a generated repair plan is done by summing up its basic costs, costs for repair operations, violated constraints, and (sub)goals. The algorithm to compute the costs is based on a plan-goal tree that starts with a single plan and attached goals and incrementally refines the goals into partial plans with subgoals. For every plan node information about its basic costs, attached subplans costs, a list of its attached goals and whether the node is a leaf is stored. Every goal node stores a list of its best plans regarding the costs of its attached subplans. Given these definition of a plan-goal tree and computing an instance of it an algorithm allows to compute the best plan considering costs by summing up the costs to the top plan node.

Differences to our approach. SEAMCAD and its respective methodology focus on establishing, respectively restoring, well-formed formulas within the models. The repair operations require inconsistencies to be present in the models. A composition of repair operations into another is not possible in the approach, in contrast to our transformation actions that allow for such a refinement of transformation actions. Furthermore, sequencing repair operations is not supported, even though Dam and Winikoff (2008, p. 220) provide sequences of repair operations. However, their sequences are considered with resolving inconsistencies in a given order, but this does not correspond to the sequence of real world changes, as formalized via transformation actions.

The plan costs in the approach only consider factors on the type level and also do not allow for an exploitation of the solution space with different resource modes. Our approach provides a ranking considering different resource modes and takes information attached to individuals into account. Furthermore, we allow for a consideration of different dimensions involved in the decision and a weighting of the different factors according to the preferences of the enterprise architect.

Our approach avoids the creation of loops within the graph grammars through the design of the transformation actions and considers the dependencies between them. In contrast the approach implemented in the SEAMCAD in prone to such loop creation and needs a loop detection mechanism (Dam and Winikoff, 2008, p. 224). Furthermore, we support top-down and bottom-up approaches.

6.2.4 Enterprise Architecture Management System

The Enterprise Architecture Management System[2] (EAMS) is a tool that allows to enrich enterprise architecture models from existing project information and the artifacts influenced by them (Sousa et al., 2009). Furthermore, it allows to detect the temporal dependencies of projects on certain elements and can generate different viewpoints that are time dependent. Time is explicitly taken into account by providing a timeline bar in

[2]http://www.link.pt/eams/DefaultEA.aspx?idl=2

the EAMS, which enables the enterprise architect to browse through the different points in time. The tool is capable of providing different states as each project has a list of elements it creates and deletes. This implies that projects are already identified and the enterprise architecture is derived afterwards.

The approach requires for every project a list of elements that are developed by it (alive list) and a list of elements that are phase out (dead list). Furthermore, a start and end date is required to determine the duration of a project and the point in time when the elements of the alive and dead list change their lifecycle phase. Therefore, EAMS is capable of showing the different intermediate architecture states based on the start and end date of each project. It is also possible to derive temporal interdependencies of projects on certain elements. However, these interdependencies are only based on the start and end date of each project and the alive and dead list of each project.

Differences to our approach. The EAMS allows to interactively create views on the enterprise architecture by using the timeline bar. However, our approach can interactively create the transformation path including alternative paths and then allows the creation of proposals for projects, their change activities and the possible synchronization between them. Therefore, our approach is more fine grained regarding the activities, i. e. transformation actions, and their start and end points that are not reconstructable from a plain list that contains elements to be developed and retired. Furthermore, we allow for decision making support in terms of a ranking that is not considered, to the best of our knowledge, within the EAMS.

6.2.5 Iteraplan for Best–Practice EA

Iteraplan[3] is a tool to be used in the Best–Practice EA approach and provides an implementation of best-practices considered by the approach (Hanschke, 2009). We focus on the planning capabilities of the tool and the methodology provided by the Best–Practice EA in the following.

The Best–Practice EA is mainly using business support maps for planning purposes. At first a current business support map is modeled to gain an overview on the current business support through applications. After a target business support map has been modeled and agreed upon a delta analysis is used to detect differences between the current and target business support maps. For a more fine grained analysis the Best–Practice EA suggests to additionally add information about application services and information objects of the applications. Afterwards, for each delta possible actions to close it are considered.

[3] http://www.iteraplan.de/

These actions range from introducing a new application, adding or reducing functionality of an existing application, changing or adding application services to the shut down of applications and application services. Based upon the results of the delta analysis and derivation of appropriate actions it is necessary to clarify dependencies between the actions, bundle the actions and create planned states as recommendations for change.

Given the different actions the Best–Practice EA suggests to bundle actions into one or more bundles. An action might be present in or more bundles and may have interdependencies with others. Furthermore, the Best–Practice EA proposes to use an evaluation schema for comparing the utility between the bundles of actions. Regarding the method used to evaluate the utility and if preferences can be set through an enterprise architect is not discussed.

Differences to our approach. Iteraplan neither allows to derive automatically the actions from identified deltas, nor reusing them for detailing a target architecture. However, the Best–Practice EA allows for considering qualitative values in the evaluation of bundles of actions. The interdependencies of actions is considered however it seems to be a manual task and becomes unclear for a large number of actions, as the approach uses a matrix with all actions on both axes. Furthermore, in which cases removing or adding an action to bundle might have side effects seems to be only dependent on the experience of the enterprise architect. A consideration of different resource modes is not considered in the tool and approach. For a detailed discussion regarding the differences between the patterns provided by the Best–Practice EA and the transformation patterns of our approach we refer to Subsection 6.4.2.

6.2.6 IAAS Cloud Cycle

IAAS Cloud Cycle is a graph based approach to improve the manageability of enterprise topologies, whereas the topology of an enterprise is the basis for modeling a current enterprise architecture (Binz et al., 2012a). A topology in this approach is a graph with edges, nodes and attributes. Edges and nodes are typed. Attributes represent information of nodes, for example the name of a node.

Via segmentation, graph transformations and analysis strategies enterprise architecture models are created. The topology is modeled manually, automatically discovered or derived from existing descriptions of applications. Segmentation is used to cluster parts of the topology that adhere to certain criteria which help to structure nodes and edges. The topology itself is the biggest segment.

Graph transformations (operations) allow to apply different reconfigurations on the topology to improve its manageability. They are used to create or remove edges and nodes and to abstract their types. The graph transformations are non-ambiguous and

backtraceable as the transformations operate directly on the topology. For applying changes that do not change the original topology, a detach transformation can be performed which creates a copy of a segment. Analysis strategies describe use cases for using the topology for deriving information. Each strategy has an objective, a problem description, a solution, and a formalization.

Strategies allow to perform an impact analysis, a workflow deep-dive, and to abstract an EA regarding the level of detail shown to the enterprise architect. Impact analysis derives all dependent nodes for a selected node, whereas the depth of the edges that are to be considered needs to be specified. Workflow deep-dive shows the dependencies of business processes via web services to nodes they are hosted on. Abstract EA allows to simplify the topology for stakeholder purposes by showing the information on a high level.

Additionally, IAAS Cloud Cycle allows for aggregating segments and filtering information shown to the enterprise architect. By aggregating a segment it is possible to merge nodes and edges of a subgraph of the topology, i. e. the segment, into a single node and edge. Filtering allows to fade out nodes and edges that have a certain type and might not be of interest for an enterprise architect in certain situations.

Differences to our approach. IAAS Cloud Cycle has with our approach the graph formalisms in common. However, it uses the graph transformation for a different purpose. IAAS Cloud Cycle aims at supporting the creation of a current architecture architecture, based on a given topology that may be derived from several different data sources. Even though the deletion and addition of nodes and edges is considered, it is not possible to represent two different states of the architecture.

Our segmentation mechanism is different to the one used in the IAAS Cloud Cycle. Our segmentation allows to derive subgraphs of current and target architecture that reduces the size of the initial states used for planning. In contrast the segmentation in IAAS Cloud Cycle creates partitions of the topology for providing models easier to handle for enterprise architects and apply transformations only on parts of the topology. As far as we can evaluate the segmentation mechanism used in the IAAS Cloud Cycle it cannot be extended to represent different states of the enterprise architecture considering changing edges. An edge is either within a segment or on the border of a segment (Binz et al., 2012a, p. 66) and thus would not allow for changing relationships between elements that are present in both.

6.3 Optimization Approaches for Planning Purposes in EAM

In this section three different optimization approaches for planning purposes in EAM are presented. Optimization in general seeks to find one or more solutions that are considered to be optimal with regard to an objective function (Sterman, 1991). Decision variables allow to determine possible solution by changing them and constraints set the limits that all solutions need to adhere to. The three approaches presented in this section vary all in the information needed for the optimization and the scope of the optimization. They all focus on the generation of an optimal state of the model and allow for a consideration of input from domain experts. However, their interactivity in finding the solutions is limited, as only the objectivity function, decision variables and constraints have an influence on the solutions. Nevertheless, such optimization approaches are crucial to create a transparency and awareness on the decision problem, besides the optimal solution they generate through the optimization technique.

6.3.1 IT Portfolio Valuation and Optimization with ArchiMate using Binary Integer Programming

The approach of Iacob et al. (2012) describes how to support IT portfolio valuation based on ArchiMate models. From the current architecture the application costs and possible consolidation costs are derived. The overall costs are the sum of consolidation costs, that are the costs that a consolidation triggers by implementing business process support to be able to shutdown another application, and application costs that are inherent to the application. Goal of the optimization is to cut the overall costs to a minimum which is done using a binary integer programming algorithm.

Before an optimization can take place an embedding of necessary information into the ArchiMate models is presented. In order to be able to model costs the approach suggests to add an attribute for costs to every element of the enterprise architecture (Iacob et al., 2012, p. 15). The goal to cut the costs and its two constituents, i. e. the goals to minimize consolidation costs and application costs, are modeled with the Motivation Extension of ArchiMate (The Open Group, 2012, Chapter 10). For the subgoals the appropriate formula on how to calculate the minimum is added to the model. Furthermore, the constraints that an application “cannot be removed if it leads to functionality loss” and that “all processes must remain operational” are added to the model. However, the formalisation of these two concepts into the binary integer programming problem is not supported later on, as it is not possible to translate natural text into a constraint of a binary integer programming problem. Furthermore, the formalization to consider data provided by application services, in order to be able to derive the target architec-

ture (Iacob et al., 2012, Figure 9), in the binary integer programming problem is not explicitly considered.

For using a binary integer programming algorithm Iacob et al. reuse the suggested approach from Franke et al. (2010), that present a binary integer programming approach to optimize the consolidation of applications and their functionality for supporting business processes. This approach creates an optimal decision on applications which should retain and be removed. Furthermore, the approach considers costs for adding functionality to applications to allow them to support further business processes that would otherwise no longer be supported due to the shutdown of other applications.

In order to be able to perform the binary integer programming optimization it is necessary to have all business processes, all abstract software functions, and a list of all applications. We assume the abstract software function to be identically to our information services, i. e. implementation independent descriptions of functionality supportable through application services that realize the functionality. Additionally, in the approach of Franke et al. it is necessary to gather the process requirements regarding the functions that need to be supported, which applications support which functions and the costs of the applications and the costs creating new functionality support through applications.

Given this information the models and attached costs are transformed into matrices that allow for determining possible solutions and the resulting costs. A valid solution is one that considers the requirements of the processes regarding abstract software functions and calculating the costs is done by adding the costs of the applications involved in the solution and the costs for implementing new functionality.

Differences to our approach. The introduction of new applications and application services is not considered. It is only possible to introduce new connections between application services and processes. Regarding the ranking in our approach we are less flexible when it comes to the individual changes. However, our approach requires less manual input as the approach from Iacob et al. where costs need to be specified for all applications, application services and possible functionality support of applications in business processes. Furthermore, our approach allows for an interactive creation of the target architecture through an enterprise architect.

As the approach from Iacob et al. only considers costs it can be regarded as a one dimensional decision problem. Even though the approach considers modeling risk elements it just considers them textually. Therefore, no weighting is considered.

Resource constraints are mentioned but not formalized and considerable in the optimization approach. In our approach we are able to provide means to consider resource constraints and also allow for a consideration of different resource modes. The only constraints considerable in the approach of Iacob et al. are processes that cannot be supported by applications.

6.3.2 Discounted Cash Flow Technique for IT Investments

A framework for assessing the cost of IT investments, considering the different costs involved in such investments is presented by Närman et al. (2009). Calculating the IT investment costs is done by gathering information of experts for different types of costs involved. The types of costs to be considered are given through a cost taxonomy. The calculation is done using the discounted cash flow technique used in financial planning to estimate the financial benefits and costs attached to investments over time (Brealey et al., 2008, p. 121). Using the discounted cash flow allows for considering incoming and outgoing flows of money resulting from a decision for different time intervals. As it is possible to consider different rates of interest for the different time intervals it is called discounted cash flow to allow for a comparison with the benefit of not conducting an investment and just depositing the investment costs at a bank.

According to the taxonomy operations and maintenance costs (1) and project costs (2) contribute to the lifecycle costs of an IT investment. The taxonomy details the costs further and even allows to consider human and organisational factors that influence the lifecycle costs. For example the 'Team system experience' or 'Post-implementation productivity loss' are considerable.

Before, the estimation can start it is an important step in the method to identify experts that are capable of providing information (Närman et al., 2009, Figure 8). Every cost to be considered is divided into cost nodes for every year that needs to be considered within the estimation of the investment. The experts then make for every cost node a estimation considering a probable future scenario and one for a very pessimistic and optimistic future.

Each of the cost nodes is weighted according to four different factors. Firstly, the experts are asked about their confidence about their estimations regarding the cost node. Secondly, the background of the expert influences the weight. Thirdly, the background factor's certainty is considered and fourthly the interviewer has to estimate the experts certainty regarding the estimation. Given these weight for each cost node a weighted average for high, probable, and low estimates is calculated. Based on this information it is possible to provide an overview on different costs for the costs nodes overall in different scenario estimations.

Differences to our approach. We consider the level of detail necessary for the estimation costs to be not realistically assessable only by enterprise architects and therefore explicitly did not call the experts involved in the estimation enterprise architects. Furthermore, the scenarios to be considered need to be defined in detail to ensure conformance between the estimations of different experts. We consider our transformation actions to be the basis for deriving such detailed scenarios.

The weights used in the approach are not comparable to our approach, as we use them to express preferences of the enterprise architect in contrast to expressing certainties. Using different resource modes for transformation actions allows to consider different time frames for the same scenario which is already predefined in the approach discussed above. Therefore, we consider our approach to be more flexible regarding the outcome of scenarios and less time consuming to gather information to conduct the decision support.

6.3.3 Optimal Distribution of Applications to the Cloud Using Topologies

An approach for supporting decisions which combination of cloud providers, backends, and databases for an application is optimal, regarding operational expenses, is presented in (Andrikopoulos et al., 2014). For each identified combination, that is considered valid, the operational costs are calculated and a partial ordering on the values is established. This allows to determine the optimal solution.

The models in the approach are called topologies and allow for typing with inheritance based on graph formalisms. Each application has a μ-topology that consists of a α-topology and γ-topology. The α-topology contains application specific information, for example if the application consists of three different components, like common for 3-tier architectures. In contrast the γ-topology considers information not application dependent, like used database and potential alternatives that could be used within the application. Inheritance in the topologies allows for a refinement in the application specific and unspecific topologies. The set of viable configurations for all applications is created through a given α-topology and γ-topology through inference based in the inheritances defined in the μ-topology.

After identifying viable configurations the ranking is created by considering the operational expenses of each component used in the μ-topology. Within the approach it is possible to add additional constraints, that however seem to just remove not viable solutions afterwards that should not have been created a priori. The only really considered constraint is the budget available that allows to restrict the configurations that do not exceed the given budget. The approach narrows the scope of the optimal decision to a single-dimension problem by only considering operational cost expenditure. Given the ranking it is possible to establish a partial order between the viable configurations.

Differences to our approach. The approach creates target architectures by modeling constraints in the μ-topology. In contrast our approach explicates the changes via transformation actions and is more general when it comes to the creation of target architectures considering larger models. The approach described above requires the modeling

of every μ-topology for every application, at least for the α-topology if no constraints are specified in the γ-topology.

The ranking in the approach for optimal distribution of applications focuses on the evaluation of the resulting state, but does not allow for a consideration of possible derivations from costs resulting from different changes. Furthermore, no influence of individual factors in the ranking is considerable, as an utility function for the optimal distribution only "depends on the types of the nodes in the γ-topology of the application" (Andrikopoulos et al., 2014, p. 82). Considering multiple criteria, as in our approach, is also not possible. The usage of the μ-topology allows for a refined consideration of potential reconfigurations within the application in contrast to our transformation actions, that are specified on type level. Our transformation actions could be either extended by means of individual decision derivations for selecting a target architecture or allow the enterprise architect to mark situations in which a transformation should not be applicable. In the latter case the transformation patterns could be checked for a necessary refinement considering individual factors and not just information available in the type graph.

The approach for an optimal distribution of applications does not allow for adding or removing nodes and edges in the graphs, which is necessary for a reconfiguration of an existing enterprise architecture in planning purposes. This is certainly a consequence of the different planning scope of our approach. Nevertheless, even an extension of the approach for an optimal distribution considering the introduction of new applications would not be possible with the given formalisms, as a priori all solutions need to be available. Even though the approach considers the budget available it is just a reduction of the search space a priori as it has not influence on the ranking of the available viable configurations. In the approach the best ranked configuration will always be the best ranked. In our approach the best ranked transformation action, in a quadruple resource mode, however might not be selectable due to resource constraints and thus impact the ranking. Furthermore, we consider our approach to be more interactive as we allow for interventions through an enterprise architect, not just in the modeling processes but also in the decisions on a micro level.

6.4 Usage of Patterns in EAM

The idea of using patterns as a means to cope with complexity inherent to architectures was introduced by Alexander et al. (1977). These pattern thinking was later adopted for designing software architectures in a model based way (Gamma, 1995). It allowed to reuse patterns that had shown their usefulness in past problems and therefore reuse model fragments for designing new software architectures. Later on the scope of the patterns was broadened from purely assisting in designing issues to analysis purposes. For

example Fowler (2001) describes analysis patterns for requirements engineering purposes related to software development issues.

Patterns have also been introduced in the EAM community. Therefore, we present two of the most prominent and mature pattern approaches for EAM and provide a distinction from our transformation patterns and transformation actions. We want to point out that there is no better or worse in this distinction. We find all pattern approaches valuable and it depends on the enterprise architects and the further stakeholders' preferences which pattern approaches are valued more or less.

6.4.1 EAM Pattern Catalog

The EAM Pattern Catalog is a methodology and set of different types of patterns that allows an enterprise to introduce and evolve EAM incrementally (Buckl et al., 2008). After setting up an initial catalog it was refined and improved with the feedback and requirements from practitioners. The EAM pattern catalog allows for a great flexibility, regarding the configuration of an EAM approach within an enterprise, by providing different combined or exclusive mechanisms that support decision making. Every pattern in the catalog has a name, a summary that allows to understand it better and an identifier. Figure 6.5 shows the different important conceptual elements of the EAM pattern catalog.

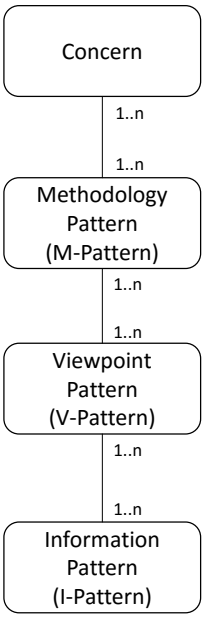

Figure 6.5: Hierarchical Structure of Concerns and Patterns in the EAM Pattern Catalog; Adapted from Buckl et al. (2008, Figure 1.3)

The concerns are the most upper hierarchical elements in the EAM pattern catalog and allow to determine the information needs of the EAM stakeholders on an abstract level. An example for a concern is "C-35: How does the application landscape look like at a specific date?" (Buckl et al., 2008, p. 33). The concerns are grouped into different topics that range from planning related concerns, to project portfolio management over to business process related concerns.

If a stakeholder decides that a certain concern is important and should be considered in the EAM approach it is necessary to determine a methodology that allows to collect the necessary information. Therefore, methodology patterns are introduced that are abbreviated by the term M-Pattern. An example of a M-Pattern is M-14 "Development of Plan and Target Landscapes" (Buckl et al., 2008, p. 64) that is a methodology pattern for C-35. The summary of M-14 is as follows:

"The M-Pattern considers the development of planned and target landscapes to support managing the evolution of the application landscape. The target landscape as a long term perspective shows the envisioned architecture of the application landscape derived from the strategies and goals of the enterprise. Planned landscapes illustrate intermediate steps, transforming the current landscape towards a target landscape. Thereby, a planned landscape shows the application landscape as it develops through the changes performed by projects up to a specific date, thus, additionally providing support for project planning." (Buckl et al., 2008, p. 64)

Each M-Pattern is in turn connected with at least one viewpoint pattern (V-Pattern) that are used within the methodology. M-14 uses four different V-Patterns: V-17, V-24, V-32, and V-40. V-Pattern V-17 is the "Process Support Map" viewpoint that allows to visualize the support of business processes in different organizational units through applications. The EAM pattern catalog points out the importance of using a legend for every view created based on a viewpoint. A view is the instance of a viewpoint that allows to support decision making, whereas the viewpoint is the generalization of the view that determines information to be visualized.

As the legend and viewpoints visualize information it is in turn important for the V-Patterns to be connected with information patterns (I-Pattern). Each I-Pattern is a fragment of a metamodel that allows to determine the information necessary to create the viewpoint it is attached to. Following a top-down approach from the concerns to the M-patterns, to the V-Patterns and attached I-patterns the corresponding enterprise architecture metamodel is configured and allows for a decision support of the stakeholders.

Differences to our approach. In the following we point out the differences between our transformation patterns (see Section 3.2) and the patterns of the EAM pattern catalog. First of all, we disregard the M- and V-Patterns, as they are considered with methodological aspects, respectively visualizations and thus are not an abstraction of

modeled situations, like the transformation patterns. However, transformation patterns are more closely related to the I-Patterns. The main difference between both is that an I-Pattern specifies parts of a metamodel that allows to create models that are instances of the metamodel. In contrast the transformation pattern is an abstraction from a modeled situations, where situations are different models, given through the metamodel, but unique in its constellation of element types and relationship types. As we use graph formalisms the uniqueness of the transformation patterns is expressed through node types and edge types. Transformation patterns could be considered as refinements of possible model instances of an I-Pattern.

6.4.2 Patterns in Best–Practice EA Approach

The Best–Practice EA approach differentiates between three different types of patterns that have different purposes (Hanschke, 2013). In general a pattern in the Best–Practice EA is a template that has proven its usefulness for certain purposes in the context of EAM. The different types of patterns are analysis pattern, design pattern, and landscape planning pattern. We will explain their characteristics in the following.

Analysis patterns assist the enterprise architect in identifying a need for action and potentials for an optimization given a current business support map. The patterns of this type allow to identify redundancies in application services, inconsistencies in data, how to realize business demands, or even technical optimization potential, like improving standard conformity platforms.

Design patterns assist the enterprise architect in defining target business support maps, after the analysis patterns have been applied to a given current business support map. These patterns allow, for example, to remove identified redundancies of application services or how to merge two business support maps from different enterprises into one. The latter pattern is used in large business transformations when one enterprise acquires another or two enterprises merge.

Possible design principles considered with this pattern are best-of-breed, one-IT, and survive. Using the best-of-breed principle selects the for each business support element the application with the highest degree of coverage regarding the process activities. Following the one-IT principle leads to one application that covers all business support elements. In contrast the survive principle does not change anything in the business support and leads to redundancies in the functionalities implemented in the applications.

Furthermore, the design patterns provide support in creating an evaluation of different designs for solutions and resulting target support maps. A schema for values to be considered in valuing the solutions is provided. The factors for the solutions to be

considered range from how much business demands are considered by the solution, to how well the solution considers the strategy (strategy fit), to the overall costs and benefits of each possible solution.

Landscape planning patterns allow to support an enterprise architect in the creation process of planned business support maps. One pattern supports the comparison between business support maps with a different localization, i. e. different coordinates or the business support map as for example different business processes are present. Such a localization can be the result of changes in the business support maps. Another landscape planning pattern allows to conduct a delta analysis to compare two business support maps for different points in time.

Differences to our approach. Transformation patterns and analysis patterns are somehow related. However, our transformation patterns are not used for analysis purposes. We use the transformation patterns to differentiate between different abstracted situations present in the models. Additionally, transformation patterns allow for a distinction between modeled situations with explicitly stating which elements or relationships are not to be allowed to be present for a transformation action to be applicable.

Transformation actions and landscape planning patterns have an interrelationship, as both aim at supporting the modeling of a possible future model. However, transformation actions have a broader scope than landscape planning patterns of the Best–Practice EA, when it comes to the support of the level of detail to be considered in the models of a future enterprise architecture. Our transformation actions allow for a determination of target architectures and not just planned business support maps. Furthermore, we are able to determine intermediate states of the enterprise architecture, i. e. planned architectures, and allow for sequencing changes of an enterprise architecture via transformation actions.

7 Demonstration and Evaluation

In this chapter we demonstrate and evaluate our approach in order to be able to achieve the fourth objective of this thesis: Create an instantiation of the approach and evaluate it (see Section 1.2). The demonstration is concerned with the creation and evolution of a transformation action repository that allows to use our approach. Furthermore, we provide a description of an instantiation of parts of our approach within a graph transformation tool. Afterwards three different use cases follow. The Living EA use case evaluates the bottom-up approach as presented in Section 4.3. Evaluating the top-down approach, as presented in Section 4.3 is done with the Best–Practice EA use case. The last use case is the Development Master Data Management use case that evaluates the transformation path generation for creating sequences of transformation actions. Afterwards, we evaluate our approach using scenarios to show its robustness and adaptability.

7.1 Transformation Action Repository Methodology and Instantiation

In this section we describe how a transformation action repository, that allows to create transformation paths interactively, is created. The usage of it was already presented in Chapter 4 and 5. Furthermore, we provide an instantiation of the transformation action repository in this section by means of a graph transformation tool that we reused as part of our approach.

We introduce a definition of the transformation action repository before we describe its methodology. According to TOGAF (The Open Group, 2011, Chapter 41) an architecture repository contains, besides other components, a metamodel for the enterprise architecture, a corresponding enterprise architecture methodology and the models for current, planned, and target architectures. We adapt this conception of an architecture repository to our approach and call it transformation action repository, as its purpose is to support the planning process of transformation paths.

Definition 32. *Transformation Action Repository: A transformation action repository contains a metamodel, transformation actions formalized via typed attributed graph pro-*

ductions and their instantiations as typed attributed graph transformations. Furthermore, a MCDM technique allows for a ranking of applicable transformation actions to provide decision support for enterprise architects. The transformation action repository is orthogonal to an architecture repository, as it intersects with the metamodel and individuals of it, but with a different purpose.

For the methodology of the transformation action repository we introduce the concepts necessary for a methodology as introduced in Subsection 2.7.2. We focus on the activities, the corresponding process, the results and techniques of the methodology. Roles involved in the methodology are not defined explicitly, as the roles necessary to conduct the activities may be instantiated differently. We point out that an enterprise architect not familiar with graph grammars is not capable to model transformation actions and different roles need to be assigned to implement the methodology. However, such a detailed role assignment is out of scope of the thesis. The metamodel of the methodology is already given through Chapter 4 and 5. A tool that supports the design and model, as well as test activities of the methodology is introduced in Section 7.1.5. Techniques for detailed guidance are just pointed out and are not discussed in detail.

Knowledge meta processes are concerned with the introduction and maintenance of knowledge-based systems (Sure et al., 2004, p. 2). Therefore, the methodology of the transformation action repository can be considered as a knowledge meta process, as its purpose is to allow for a creation and maintenance of the transformation action repository. Figure 7.1 shows the high-level process for the transformation action repository methodology.

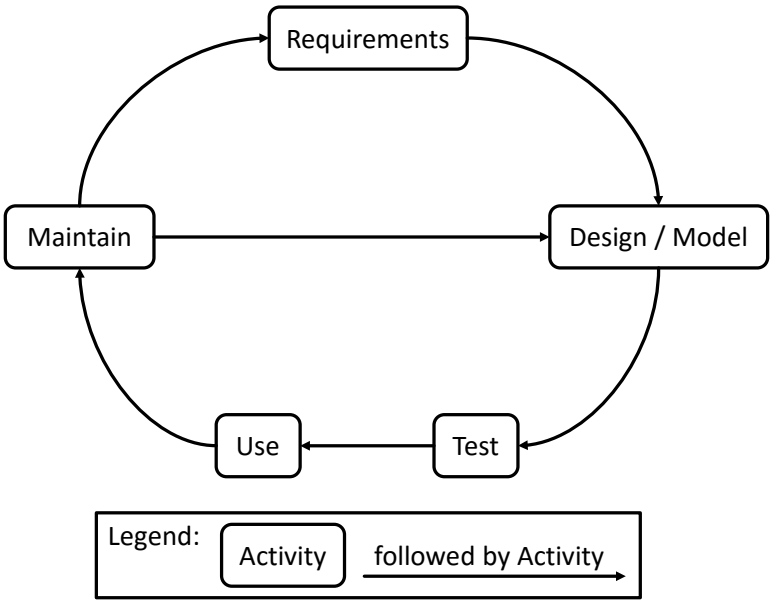

Figure 7.1: High-level Process of the Transformation Action Repository Methodology

The process is an adapted version of the unified software development process introduced by Jacobson et al. (1999). We did not depict that insights from a later activity might require the reiteration of a former activity due to gained insights. However, it should be clear that for example issues detected in the test phase might require a remodeling of the transformation actions.

Overall, the process of the transformation action repository methodology consists of five different activities. Initially, the process starts with the requirements elicitation for the transformation action repository. This activity is followed by the design and modeling of the transformation actions and their ranking. Afterwards, the transformation actions, their ranking and the selection processes are tested. If the tests are passed successfully the transformation action repository is ready for use. During the usage activity further improvements for the transformation action repository might be issued that need to be considered in the maintenance activity. The latter is concerned with directly influencing the design and model activity as well as it might trigger additional requirements that need to be considered in a future version of the transformation action repository.

7.1.1 Activity Requirements

The first activity is concerned with gathering requirements for transformation actions to be modeled and how they are ranked to provide decision support to the enterprise architect later on.

7.1.1.1 Requirements for Transformation Actions

We used two different techniques to gather information from enterprise architects that we formalize as transformation actions. On the one hand we asked for transformation actions the enterprise architect wants to consider. On the other hand we provided suggestions for transformation actions derived from the enterprise architecture metamodel. All discussions should be based on real world examples to allow the enterprise architect to express her information needs in issues she is familiar with. The knowledge engineer is concerned with the abstraction and formalization of the requirements described by the enterprise architect. It is necessary to determine the preconditions and effects the transformation actions should have. Furthermore, it is necessary to determine which modeled information would prohibit the execution of transformation actions and what information is considered to be inconsistent and is not allowed to be created by the automated planner. The latter would be for example the case if an application uses an application service it implements.

The knowledge engineer also needs to determine what the start and goal states look like to support the activities in target architecture selection and transformation path generation. Additionally, it needs to be decided whether the goal states are to be represented explicitly or in terms of conditions a goal state has to satisfy. This allows to mark goal states and reachable goal states by an automated planner.

If the planning domain model does not consider elements, relationships or attributes that an enterprise architect wants to consider the planning domain model needs to be extended. Furthermore, it needs to checked whether this information is already representable in the enterprise architecture metamodel. If this is not the case it needs to be checked what the efforts are to gather the necessary information in an appropriate quality.

7.1.1.2 Requirements for Ranking Transformation Actions

To be able to rank the transformation actions it is necessary to determine the values that are to be considered in the ranking and the preferences need to be discussed and initially set. The preferences should be discussed based on the real world examples and some fictive examples that allow later on to determine the consistency in the preferences. If factors that should be considered in the ranking are not derivable from model instances of the planning domain model it needs to extended.

7.1.2 Activity Design and Model

The knowledge engineer models the transformation actions as typed attributed graph productions and creates the tables needed for ranking. Furthermore, she creates typed attributed graph productions that may be necessary to be run in the background and are relevant for the automated planner, but have no relevance for the enterprise architect. Referring back to our approach the typed attributed graph productions shown in Figure B.1 to B.11 represent such formalisms. Furthermore, queries to extract the initial state and a transformation of certain information into typed attributed graph productions for goal states needs to be elaborated by the knowledge engineer. An appropriate query model query language needs to be available. In our approach we used SPARQL, as we reused models based on OWL.

7.1.3 Activity Test the Transformation Action Repository

In this phase the knowledge engineer tests the modeled transformation actions and consistency of ranking. Furthermore, typed attributed graph productions should be modeled to detect states that are not allowed to be created during transformation path planning. The testing should be supported by automatic means to search for modeling inconsistencies. This allows for example to determine if all selection sequences of transformation actions lead to goal states and if during the selection process typed attributed graph productions are available that are forbidden. We call the latter debugging actions that allow us to detect such forbidden states. Additionally, it is necessary to test the results of the ranking and adapt the values of the MCDM technique in use according to the enterprise architect. A calibration of the values in the ranking, provided by the enterprise architect, may be necessary if the created suggestions do not reflect her preferences.

7.1.4 Activity Maintain the Transformation Action Repository

Maintaining the transformation action repository is necessary to allow for the adaptation of transformation actions and rankings. Furthermore, it should be possible to deactivate transformation actions and introduce new ones. Therefore, it is recommended to provide a catalogue for the transformation actions that allows to determine which transformation actions are alternatives to each other and if they are a composition of each other. Goetz and Maurer (2011) provide a good overview on how such a catalogue is created and used in the context of application integration. The maintenance may also trigger an extension of the planning domain model to allow further transformation actions to be modeled or consider further information necessary for ranking. Using a catalogue for the transformation actions allows also to create compact views on transformation actions that are alternatives to each other, transformation actions that have interdependencies, or if the transformation action is a composition of several others. This helps the knowledge engineer and enterprise architect to gain an overview on the transformation actions and estimate impacts of future changes to the transformation action repository.

7.1.5 Transformation Action Repository Instantiation

In the following we address the instantiation of the parts of our approach concerned with graph grammars. Therefore, we did a literature review regarding comparisons of graph transformation tools that could be reused for our purposes. A good overview of existing tools and their capabilities is given in Ghamarian et al. (2012, Table 6). We focused on the general purpose tools and as criterion within the remaining we decided to choose the one with the advanced rule features for quantification.

This tool is called GRaphs for Object-Oriented VErification (GROOVE[1]). Besides, providing the capability to "explore the entire state space generated from different rule application sequences" (Ghamarian et al., 2012, p. 38) GROOVE has also shown its applicability to automated planning problems (Estler and Wehrheim, 2011; Edelkamp and Rensink, 2007).

The current version of GROOVE allows to import and export Ecore[2] conform models, which can be for example Unified Modeling Language (UML) models. As we reused models based on OWL we use SPARQL queries to extract the models necessary for transformation planning and transform them into GROOVE models. The latter are based on the Graph Exchange Language (GXL) (Holt et al., 2006) that is a Extensible Markup Language (XML) variant for graphs.

Given the enterprise architecture metamodel and the planning domain model, in its respective syntax, we specified a mapping between the metamodels to allow for an automatic transformation from models based on one metamodel into a model corresponding to the other metamodel.

GROOVE provides a user interface to model the transformation actions, but also for modeling initial states and the type graph. Furthermore, it provides sophisticated means for state space exploration that allows to easily generate possible changes to modeled states automatically. However, GROOVE is not well suited for knowledge management purposes regarding enterprise architecture management issues, as it requires knowledge on graphs and graph transformations. Therefore, we integrated GROOVE in the background to allow for a seamless change of the formalisms in the background without the enterprise architect noticing it. Additionally, we had to integrate the ranking mechanisms and provided an overview of currently selected transformation actions for the enterprise architect.

Regarding the limitations of GROOVE we had to lower the requirements for the formalisms. In GROOVE it is possible to use a type graph, however attributed type graphs are not supported. Furthermore, GROOVE uses a single pushout approach for graph transformations, that is less expensive in computational time, but more prudence in modeling the transformation actions and testing side effects is necessary. For example the identity condition is not checked by GROOVE as long as no isomorphic match is enforced through GROOVE. It is either possible to activate the isomorphic match in GROOVE or explicitly enforce the inequality of nodes by using a **!=**-operator.

[1] http://groove.cs.utwente.nl/
[2] http://www.eclipse.org/modeling/emf/?project=emf

7.1.6 Summary on Transformation Action Repository

In this subsection we presented the methodology for a transformation action repository and instantiation of parts of it within the graph transformation tool GROOVE. The presented methodology is not limited to incremental and iterative development approaches. However, we advise to use such approaches as they provide the benefit of fast feedbacks from enterprise architects and allow for a managed evolution of the transformation action repository. A technical decision against or for a certain graph transformation tool should be based on the required formalisms and the scalability of the tool. GROOVE has already shown its scalability for automated planning problems with state spaces up to two million states (Edelkamp and Rensink, 2007, Table 1), even though the computation time is high for such large state spaces. We expect the state spaces to be taken into consideration by enterprise architect's far less big. Furthermore, we point out that current research on pattern abstraction of state spaces shows that it is possible to reduce the state space by abstracting the graphs states to its patterns and thus allows to converge different graph states, sharing the same patterns, into abstracted graph states (Zambon, 2013).

7.2 Living EA - Use Case

The following use case shows the bottom-up approach for decision support in selecting a target architecture as introduced in Section 4.2. It is called the Living EA - Use Case, as the methodology and models, that were used to obtain the results, were created using the Living Enterprise Architecture approach[3]. The use case itself is from a small consulting enterprise's project handling process that is to be improved by means of changing IT support in the process activities.

7.2.1 Preliminaries for the Living EA - Use Case

As a starting point the enterprise wants to change the IT support for its process *Customer Project Handling*. The drivers for the change are automating the process as far as possible and a demand for replacing old application services. The process *Customer Project Handling* was refined into its process activities (x-axis of Figure 7.2) starting with the *project planning* activity and ending with the activity for *project closure*.

Figure 7.2 shows the modeled current and target business support map. Application services and one information service, highlighted by a star symbol, where modeled by

[3]http://www.living-enterprise-architecture.de/

an enterprise architect in the business support maps. The application that implements the respective application service is depicted in brackets. Besides the general manager only the role of the project manager is involved in the process. Within the current business support map the process activities *project planning* and *project permission* do not localize an application service via their business support elements.

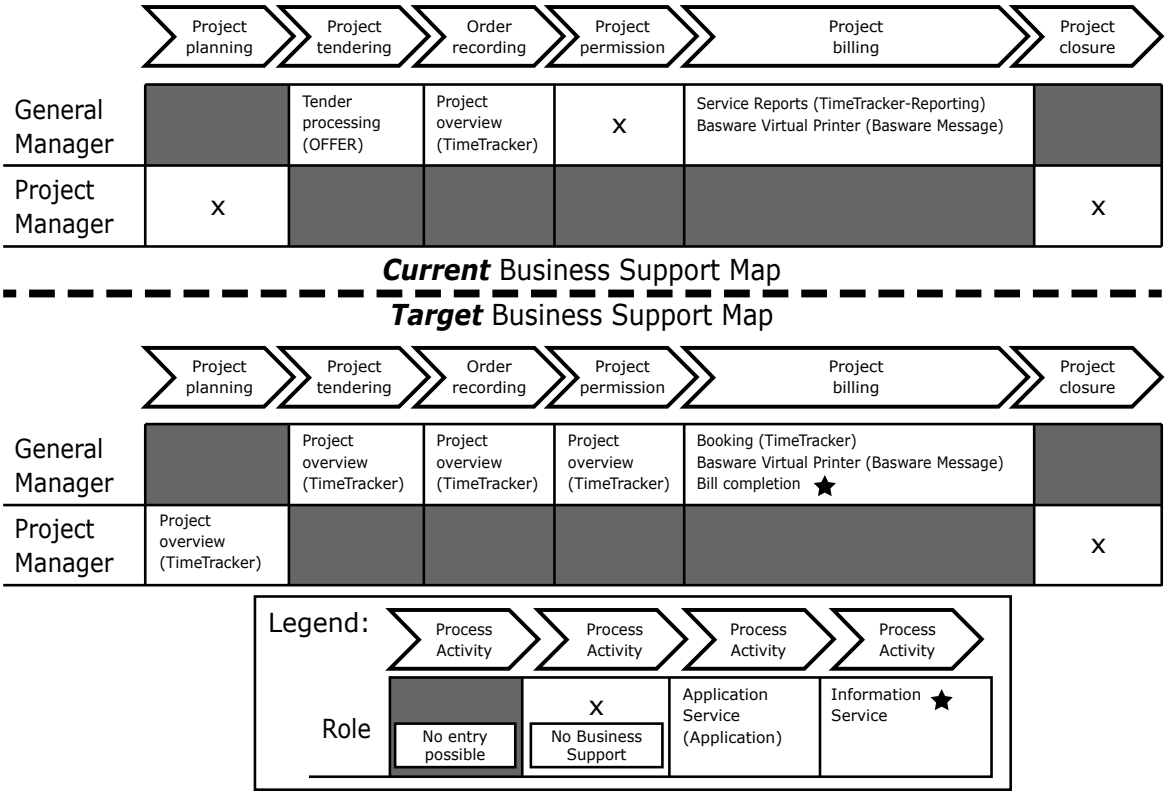

Figure 7.2: Current and Target Business Support Map for the Process *Customer Project Handling* in the Living EA - Use Case

The process activity *project billing* localizes two application services via its business support element. Each process activity in the target business support map localizes at least one application service. For example, the *project tendering* activity in the current business support map is supported by the *tender processing* application service, which is implemented by the application *OFFER*. The application service is to be replaced by the application service *project overview* implemented by the application *TimeTracker*.

For the process activity *project billing* an information service was specified. No application realizes this information service via an application service it implements. The information service for *bill completion* updates the information object *Bill* which has five schema entries. From Table 7.3 we derive the effort baseline $PM = 1$.

For the modeled information service *bill completion* four alternative transformation actions are applicable. The first alternative allows to reuse the *booking* application service as it already realizes an information service which uses the information object *Bill*. All the other transformation actions can be executed in the different modes of resource allocation. The second alternative is the functional enhancement of the application service *service reports* implemented from the *TimeTracker-Reporting* application. Thirdly, it is possible to realize the information service via a new application service through an existing application. In the use case the applications *TimeTracker-Reporting* or *TimeTracker* could implement such a new application service. As a fourth alternative it would be possible to create a new application which implements the necessary application service.

Given this information it is possible to create the initial state S_{BU}. Figure 7.3 on page 148 shows S_{BU} for the Living EA use case. Besides, the information shown in the business support maps of Figure 7.2 the initial state contains also all information services and information objects for the modeled application services. S_{BU} contains thirty-five graph nodes (V_G) and fifty graph edges (E_G).

Additional Information for Ranking

In the following we introduce the different values we gathered from an enterprise architect to calculate the ranking of the transformation actions.

Given three dimensions cost, time, and risk the enterprise architect decided that the first dimension is as important as the two others together. However, the remaining weight was not considered as equally important. The time dimension turned out occupy seventy percent of the remaining weight in contrast to the risk dimension. The resulting weights w_j are shown in Table 7.1.

Weight Dimension	Weight Variable	Weight Value
Cost	w_1	0,5
Time	w_2	0,35
Risk	w_3	0,15

Table 7.1: Weighting for the Different Dimensions in the Living EA - Use Case

We proceed with the description of the values used for the risk dimension. Table 7.2 shows the assignment of the risk values to transformation actions. Reusing and replacing was considered to be equally risky by an enterprise architect, as they result in the same change. That the replacement of an existing application service could be considered as more risky than just the addition of a new application service was not considered by her. Therefore, no differences between the transformation actions of with and without the IS-Suffix exist for the risk dimension.

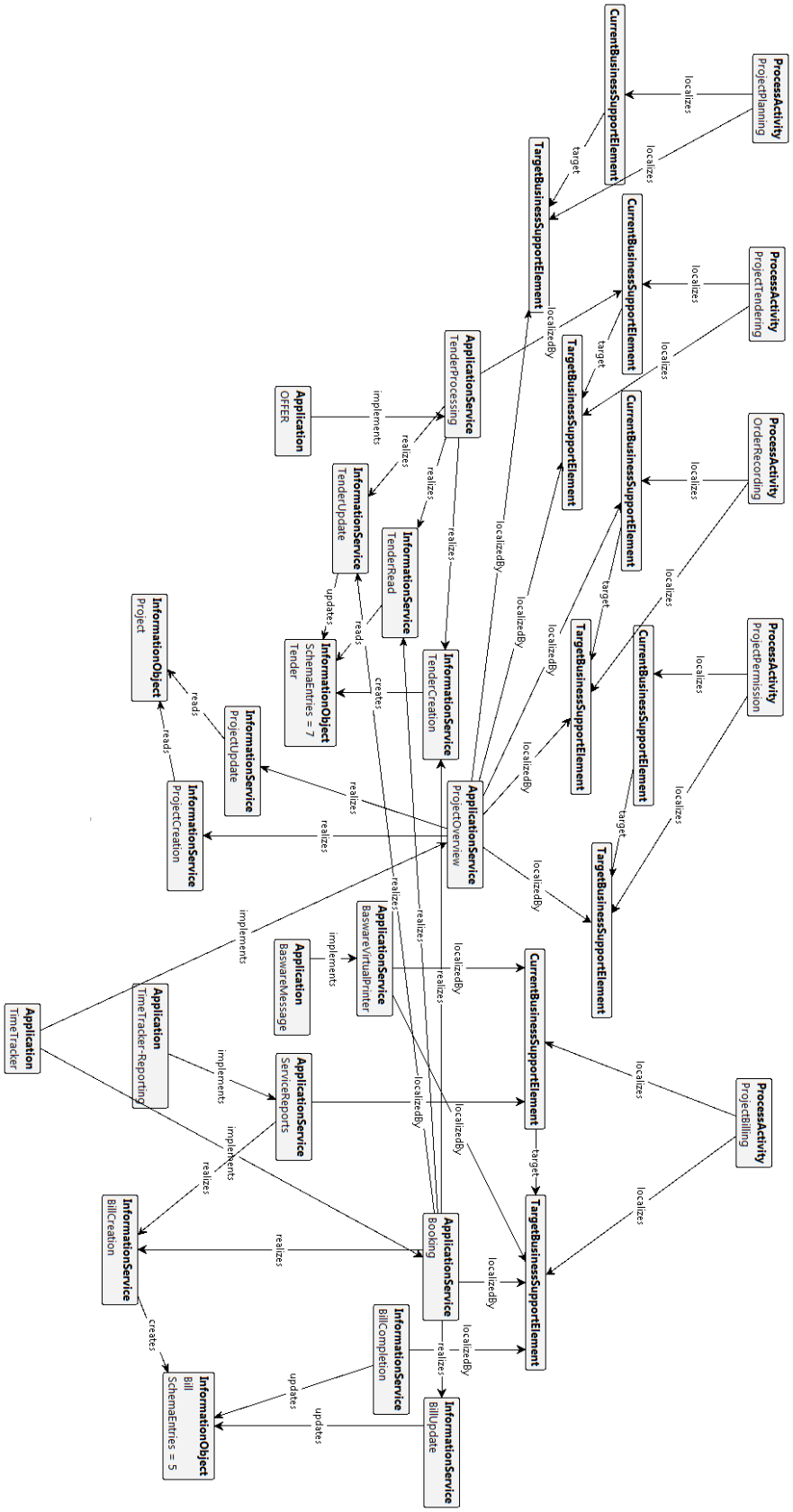

Figure 7.3: Initial State S_{BU} for the Living EA - Use Case

However, enhancing an application service is more risky than reusing or replacing one. The next risky transformation actions are those that develop a new application service but reuse an existing application. Developing a new application service with a new application are considered as the most risky transformation actions. Please note that we used already normalized values for this dimension.

Transformation action	Risk Value Z_P
$P_{ReplaceAS}$	0,1
$P_{ReuseAS}$	0,1
$P_{EnhanceAS}$	0,4
$P_{NewASOldA}$	0,8
$P_{NewASNewA}$	1
$P_{ReuseAS_{IS}}$	0,1
$P_{EnhanceAS_{IS}}$	0,4
$P_{NewASOldA_{IS}}$	0,8
$P_{NewASNewA_{IS}}$	1

Table 7.2: Assignment of Risk Value Z_P to Transformation Actions in the Living EA - Use Case

In order to be able to calculate the variable values in the time and cost dimensions person-month as a unit for creating effort baselines were introduced. Table 7.3 shows the person-month efforts depending on the schema entries for an information object. Information services that are to be realized through application services and create, read, update, or delete an information object with one to five schema entries result in an effort of one person-month. Six to five schema entries result in one and a half person-month effort and all information objects with eleven or more schema entries result in two person-month efforts.

Information object schema entries	Effort baseline in person-month (PM)
1-5	1
6-10	1,5
11++	2

Table 7.3: Information Object Schema Entries and Corresponding Effort Baseline in Person-Month PM in the Living EA - Use Case

The values for the costs have been determined by the enterprise architect as fixed costs C_f. Developing a new application service results in additional costs C_s and C_a if an application is developed. Variable costs C_v allow then to calculate the variable costs

depending on the schema entries of the information object. Table 7.4 shows the cost values.

Cost Value Dimension	Cost in Euros
C_f	2.500
C_a	7.500
C_s	2.500
C_v	2.500 $\times PM$

Table 7.4: Assignment of Costs to Cost Value Dimensions for the Living EA - Use Case

As, we allow for the application of transformation actions in different resource modes we had to determine the scaling factors with the enterprise architect. Table 7.5 shows the setting of the scaling factors chosen by the enterprise architect for scaling the cost and time dimension, that allow to calculate different efforts depending on the resource mode. In this case the values for scaling down time and scaling up costs are proportional inverse and were derived based on experiences of former projects.

Scaling factor	Value
c_q	1,2
c_d	1,1
t_q	0,3
t_d	0,55

Table 7.5: Scaling Factors for Different Resource Modes in the Living EA - Use Case

7.2.2 Interactive Target Architecture Selection for the Living EA - Use Case

In the following we introduce the transformation actions applicable in the Living EA use case. The transformation actions that can be applied in different resource modes are just listed once below. $R_{P_{ReuseAS}}$ is not applicable as no application service currently realizes the information service *bill completion*. We grouped the list by transformation actions and the participating individuals below them. In brackets behind the transformation actions is the reference to the typed attributed graph production and the number of transitions in GG_{BU} created through it. The TA with a number before an applicable transformation action is used to reference it in the ranking that we introduce later on.

- Replace application service ($P_{ReplaceAS}$ eighteen):

  - $TA1$ Tender processing (OFFER) replaced through application service *project overview* (TimeTracker)

  - $TA2$ Service reports (TimeTracker-Reporting) replaced through application service *booking* (TimeTracker)

- Enhance existing application service ($P_{EnhanceAS_{BU}}$ twelve):

  - $TA3$ Service reports (TimeTracker-Reporting) to realize information service *bill completion*

- Develop new application service with existing application ($P_{NewASOldA}$ twenty-four):

  - $TA4$ New application service AS1, to realize information service *bill completion*, implemented through TimeTracker

  - $TA5$ New application service AS1, to realize information service *bill completion*, implemented trough TimeTracker-Reporting

- Develop new application service with new application ($P_{NewASNewA}$ twelve):

  - $TA6$ New application A1 and new application service AS1 that realizes the information service *bill completion*

Figure 7.4 shows the different states reachable through the application of transformation actions on S_{BU}. Overall, GG_{BU} creates twenty-five states and seventy-one transitions, whereas five states (s16, s21, s22, s23, and s24) and the transitions to them can be neglected, as the application of $P_{goalState_{BU}}$ adds no information that is relevant for the enterprise architect. Different resource modes for the transformation are considered in the transitions between the states, but result in the same state. Therefore, each transition edge in Figure 7.4, representing a transition for a transformation action that is executable in different resource modes, counts for three transitions. One exceptional issue about the selection process is that if the application service *Service reports* is enhanced via a transformation action, it cannot be longer replaced by the *booking* application service implemented by the application *TimeTracker*, as the application service to be replaced is now localized by a target business support element.

Table 7.6 shows the ranking for the applicable transformation actions. The lower a value for the ranking of a transformation action is the better the application of it is considered to be, as we calculate a ranking of their efforts. Ranking values for each applicable transformation action correspond to the edge weights of the transformations in GG_{BU} as shown in Figure 7.4. As, no transformation action that reuses an existing application service is present, the best ranked transformation actions are $TA1$ and $TA2$ for replacing different application services. Given the weighting from Table 7.1, where costs are more important than time, does not imply that the transformation actions in normal mode are ranked better than those in the quadruple mode. Therefore, every transformation action in its higher resource mode is ranked better than the one in the lower resource

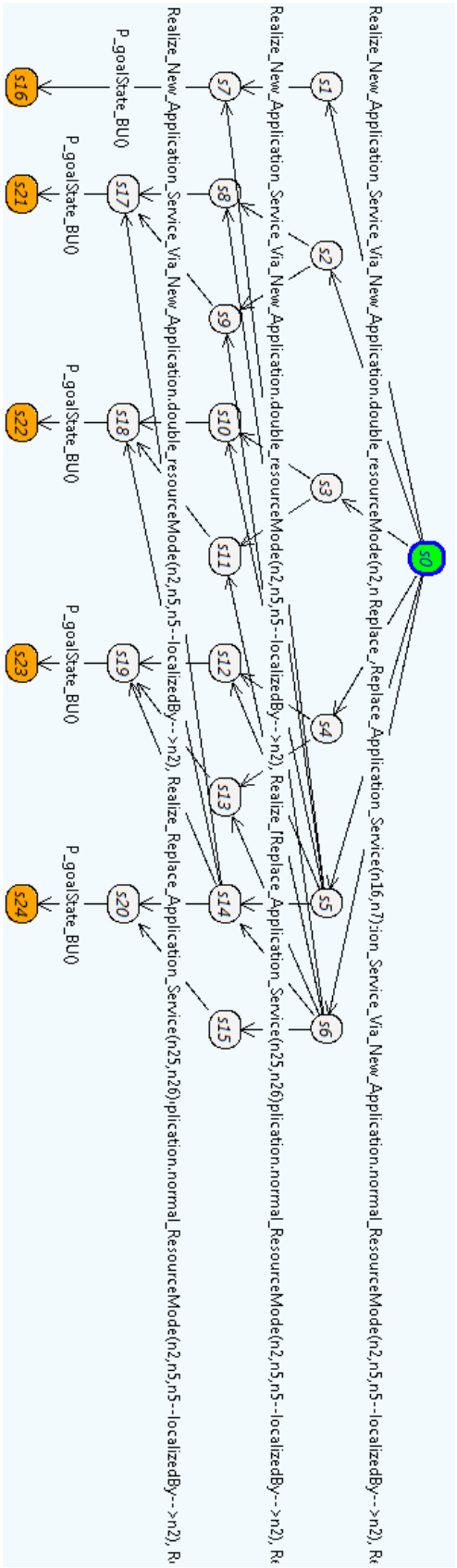

Figure 7.4: Possible Target Architectures Generated Through the Graph Grammar GG_{BU} for the Living EA - Use Case

mode. This is because the extra costs, a higher resource mode causes, are only a part of the costs. The fixed costs are the main part, given through the functionality to be implemented, and therefore a faster and costlier realization of the change is considered as better than a slower and cheaper realization.

Transformation Action	Ranking Value
$TA1$	63,80364657
$TA2$	63,80364657
$TA3$ in quadruple mode	70,85500235
$TA3$ in double mode	73,62151408
$TA3$ in normal mode	78,55150302
$TA4$ quadruple mode	94,81753592
$TA5$ quadruple mode	94,81753592
$TA4$ double mode	99,24998360
$TA5$ double mode	99,24998360
$TA4$ normal mode	106,74674180
$TA5$ normal mode	106,74674180
$TA6$ quadruple mode	166,63667520
$TA6$ double mode	171,36573380
$TA6$ normal mode	179,90286050

Table 7.6: Ranking Values for Different Resource Modes in the Living EA - Use Case Ordered by Ranking Value

If the transformation actions were to be chosen automatically the list of transformation actions $PD_{selected}$ would contain $TA1$, $TA2$, and $TA3$ in quadruple mode. However, the enterprise architect decided to select $TA1$, $TA2$, and $TA4$ in normal mode. The derivation between both selections originates from the fact that the realization of the bill completion service is not considered as urgent and has a minor importance to the process owners, i. e. the general managers and project managers. The enterprise architect knew about this and therefore considered it in the selection process. Furthermore, she decided to select transformation action $TA4$ instead of $TA5$, as she considered the application *TimeTracker* to be more sustainable than *TimeTracker-Reporting*. For the other business processes no changes were planned. Besides, the transformation actions and their ranking, G_{goal} was created.

We now finalize G_{goal} to create EAG_{target}. The application services *project overview*, *booking*, and *basware virtual printer* are set to stable automatically with $P_{finalize_{BU1}}$. As, no application services, that are not localized by a business support element, were present in S_{BU} the graph production $P_{finalize_{BU2}}$ is not applicable. $P_{finalize_{BU3}}$ sets *tender processing* and *service reports* to the lifecycle phase live. Then the applications *TimeTracker* and *Basware Message* are set to stable via $P_{finalize_{BU4}}$. Afterwards, *OFFER* and *TimeTracker-Reporting* are set to lifecycle phase live via $P_{finalize_{BU5}}$. As, we did not create any cycles in the transformation model for the application services during

the selection process we cannot apply $P_{finalize_{BU6}}$ and $P_{finalize_{BU7}}$. However, as we did not add any successor relationships for the applications so far we apply $P_{finalize_{BU8}}$ to add the successor relationships for the applications. *TimeTracker* is now a successor of *OFFER* and *TimeTracker-Reporting*. We did not create any cycles in the transformation model for the applications and therefore we cannot apply $P_{finalize_{BU9}}$ and $P_{finalize_{BU10}}$. As, the documentation on all existing usage dependencies was not available we were not able to verify the change of the usage dependencies between application services and applications. Therefore, we decided to take the created graph as EAG_{target} and we could proceed with the transformation path generation as described in Chapter 5.

7.2.3 Summary of the Living EA - Use Case

We demonstrated the bottom-up approach for supporting decisions involved in selecting a target architecture with the Living EA use case. Besides, showing how the information modeled by an enterprise architect is reused within the graph grammar GG_{BU} we exhibited the creation of transformation actions and their ranking. Furthermore, we explained how derivations between the transformation actions suggested by the ranking and the actual selected transformation actions have emerged. We were not able to verify the change of the usage dependencies between application services and applications as the documentation on all existing dependencies was not available. Nevertheless, we showed that the selection process is transparent to an enterprise architect and that it is possible to create an EAG_{target}.

7.3 Best–Practice EA - Use Case

The Best–Practice EA use case demonstrates the top-down approach as presented in Section 4.3. It is called Best–Practice EA use case, as it stems from the EAM approach developed and described in detail in Hanschke (2009, 2013), that uses this name. We decided to reuse the detailed descriptions provided by the books and the extra materials (Hanschke, 2012), as a starting point with publicly available information, and show the appropriateness of our approach to other EAM approaches.

As, we cannot provide any insights from a real world use case here we describe the Best–Practice EA use case on a neutral level. This a consequence of the fact that we reuse the information provided in the sources of Hanschke and had no industrial partner that used the Best–Practice EA approach. Where necessary we justify the assumptions we had to make to reproduce parts of a decision making process for this use case.

Before we start to describe the details of the use case we point out two significant characteristics regarding the metamodels of the PDM and the one provided by the Best–Practice EA (Hanschke, 2013, p. 204). Firstly, even though the metamodel of the Best–Practice EA considers information objects as entities (see Table 3.1), it does not consider information services as used in the PDM. The application services, in the Best–Practice EA called interfaces, directly access the information objects. As, we have the enterprise architecture modeled in a knowledge base that allows us to determine inconsistencies and implicit information we add the necessary information automatically. This is possible because we know which application services access which information objects. So we automatically add for each access connection an information service (entity) that uses (relationship) the information object and is realized (relationship) by that application service. This can be done using SPARQL as simple rule language for semantic web based knowledge bases (Allemang and Hendler, 2011, p. 88 ff.). Therefore, we can ensure the structure defined through the PDM, but without additional manual modeling efforts.

Secondly, the Best–Practice EA metamodel does not provide a logical application element. There are two options regarding the general applicability of our approach to this issue. On the one hand the enterprise architect might decide, that she does not want to use the logical applications as a frame setting mechanism in the top-down approach. The transformation action repository is not influenced by this issue, as the applicability of the transformation actions is ensured. However, the only transformation action applicable in the first phase of the top-down approach would be $P_{ReplaceApp}$. On the other hand, the enterprise architect may decide to use logical applications. As a result for every application, that she wants to get the additional transformation actions provided for, a logical application has to be specified. Even if not for every application a logical application is determined our approach remains usable.

Given these differences and their consequences we have to make assumptions how logical applications could be used within the Best–Practice EA approach. In a real world use case with enterprise architects at hand, we could have just asked which adaptations to the approach should be applied. However, for the Best–Practice EA use case this was not possible. Therefore, we will exemplify and justify our assumptions in the following.

7.3.1 Preliminaries for the Best–Practice EA - Use Case

In this subsection we introduce the models used for the interactive decision support in target architecture selection and necessary input for the ranking. The current and target business support map shown in Figure 7.5 is adapted from Hanschke (2012, p. 43). Both business support maps allow for the same localization of applications, via business processes and organizational units. On the x-axis the two business processes *Customer Management* and *Order Fulfillment* are depicted in Figure 7.5.

The organizational units *Headquarters*, *Subsidiary Munich*, and *Subsidiary Cologne* form the y-axis of the business support maps. The cells of the business support map shown in Figure 7.5 are populated with the applications. For example, the application *Total CRM* is used in the *Headquarters* for *Customer Management*. The remaining applications are related to the organizational units and business processes as shown in the Figure.

Please note that we named the entities in our business support maps differently than given in Hanschke (2012, p. 43). We did this as the original source provides just enumerated 'dummy' units for the organizational units. Furthermore, we had to translate German terms given in Hanschke (2012, p. 43; Abbildung B.36) and named the applications differently to avoid the break of copyright laws.

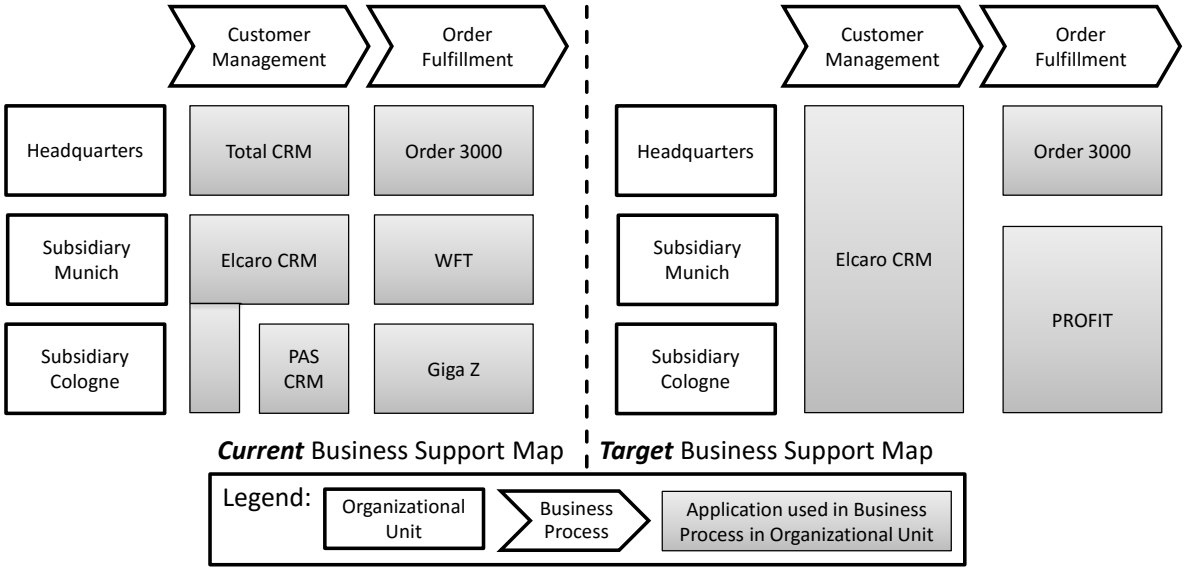

Figure 7.5: Current and Target Business Support Map on a Value Chain Level for the Best–Practice EA - Use Case

As already mentioned no logical applications are considered in the examples provided in Hanschke (2012). Therefore, we will explain how they could be modeled and considered in the target business support maps. Figure 7.6 shows the consideration of logical application for the Best–Practice EA use case. As we see in Figure 7.5 the application *Elcaro CRM* is used in the subsidiaries, but not in the headquarters in the business process *Customer Management*. In contrast in the target business support map it is used within all organizational units. If we would model a logical application *Customer Relationship Management Application*, that is an implementation of *Total CRM*, *Elcaro CRM*, and *PAS CRM*, and that is used in all three organizational units for *Customer Management*, it is possible to derive the situation depicted in Figure 7.5 by reusing *Elcaro CRM*. The same holds for the applications *Giga Z* and *WFT*, that are implementations of the logical application *Order Management Application* and *PROFIT* is the name of a newly created application.

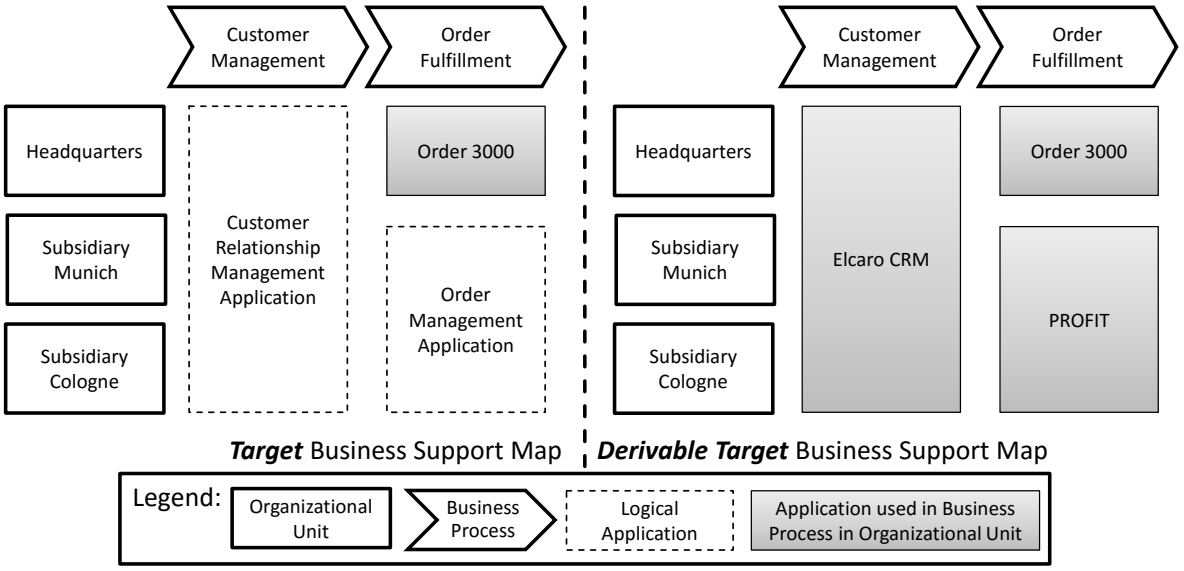

Figure 7.6: Target Business Support Map and a Derivable Target Business Support Map as Shown in Figure 7.5

Besides, the information in the business support maps, we modeled application services, information services, and information objects. Therefore, we are able to determine the functional deltas between the applications and allow for an interactive decision support in the second phase of the top-down approach. Figure 7.7 on page 158 shows the initial state S_{TD}.

We assume the application *Elcaro CRM* to implement the most functionality by realizing the most information services for the information object *Customer*. *PAS CRM* and *Total CRM* implement the same functionality, as they only allow for the creation of Customers. The order management applications *Giga Z* and *WFT* implement different functionality for the information objects *Bill*, *Order*, and *Contract*. For the application *Order 3000* we did not model this detailed information, as no transformation actions will be available for the transformation pattern (ii) that is present here in the business support maps.

Given the initial state we derive the relevant functional deltas between the applications. Table 7.7 shows the functional deltas for the Best–Practice EA use case. The functional delta between *Giga Z* and *WFT* is as high as the vice versa functional delta of them. This is because *Giga Z* realizes two information services not realized by *WFT* and *WFT* realizes two information services not realized by *Giga Z*. As, *Elcaro CRM* realizes all information services that *PAS CRM* and *Total CRM* realize its functional delta to both is zero. In contrast the functional delta from *PAS CRM* and *Total CRM* to *Elcaro CRM* is one, as they both do not realize the information service *Customer Solvency Check*. The functional delta between *PAS CRM* and *Total CRM* is in both directions zero as none of them does realize an information service that the other does not.

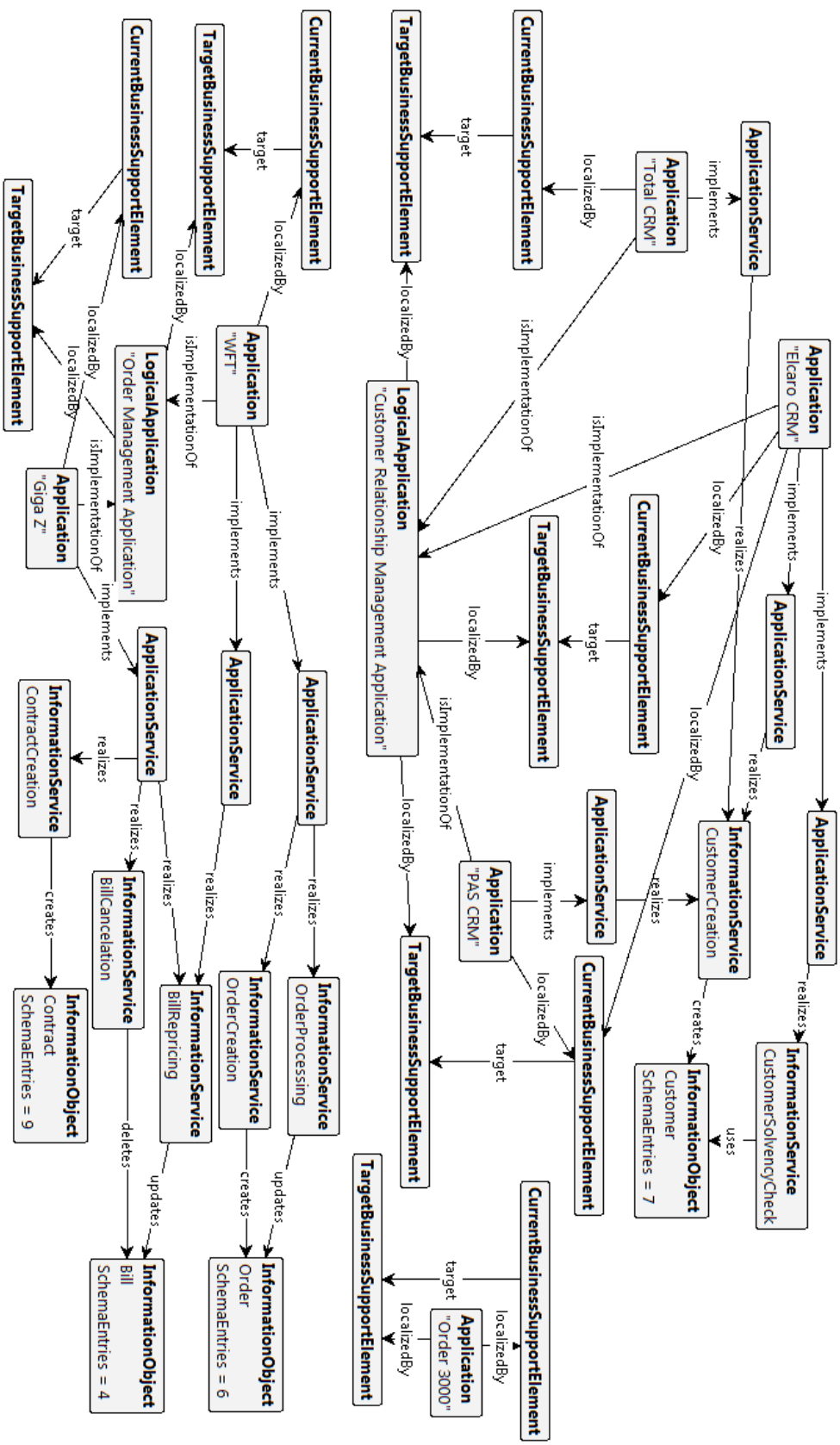

Figure 7.7: Initial State S_{TD} for the Best–Practice EA - Use Case

Application 1 and Application 2	Functional Delta
Giga Z, WFT	2
WFT, Giga Z	2
Elcaro CRM, PAS CRM	0
Elcaro CRM, Total CRM	0
PAS CRM, Elcaro CRM	1
Total CRM, Elcaro CRM	1
PAS CRM, Total CRM	0
Total CRM, PAS CRM	0

Table 7.7: Functional Deltas for the applications in the Best–Practice EA - Use Case

Additional Information for Ranking

Besides, the business support maps and the underlying information about usage dependencies of application services, it is necessary to determine the values to allow for a ranking of the transformation actions. We reconstructed the values, as far as possible, from the examples provided in Hanschke (2009, 2013). Additionally, we had to make subjective choices in determining the values as, for example, no values for the weighting of the dimensions was reconstructable. However, we argue that every decision made is prone to subjectivity and that the validity of the approach is not influenced by the values used in the ranking, as they may be adjusted to create acceptable results for another enterprise architect.

We now continue with the description of the different values used for weighting. Table 7.8 shows the weights for the cost, time, and risk dimension. Within the weighting the cost dimension is considered to be the most important dimension with almost the half of the weighting. The time and risk dimension are also considered as important. However, time is regarded to be more important than risk.

Weight Dimension	Weight Variable	Weight Value
Cost	w_1	0,45
Time	w_2	0,3
Risk	w_3	0,25

Table 7.8: Weighting for the Different Dimensions in the Best–Practice EA - Use Case

Table 7.9 shows the different values for the risk of each transformation action, whereas the risk values for $P_{EnhanceAS_{TD}}$ and $P_{NewAS_{TD}}$ are used in the second phase for the

detailed decisions. However, these values influence also the risk in the first phase for calculating the Min, Max frame (see Subsection 4.3.2). If the functional delta is zero the calculated effort would be also zero. However, this would not be a realistic estimation for the transformation actions with such functional deltas. Therefore, we introduce a non-variable value Z_{def} for the risk when the functional delta is zero.

Transformation action	Risk Value Z_P
$P_{ReuseApp_{min}}$	Functional Delta × 0,45
$P_{ReplaceApp_{min}}$	Functional Delta × 0,45
$P_{NewApp_{min}}$	0,75 + (Functional Delta - 1) × 0,45
$P_{ReuseApp_{max}}$	Functional Delta × 0,75
$P_{ReplaceApp_{max}}$	Functional Delta × 0,75
$P_{NewApp_{max}}$	Functional Delta × 0,75
Z_{def}	0,15
$P_{EnhanceAS_{TD}}$	0,45
$P_{NewAS_{TD}}$	0,75

Table 7.9: Assignment of Risk Value Z_P to Transformation Actions in the Best–Practice EA - Use Case

We continue with the assignment of person-months to schema entries to allow for a calculation of variable costs and time. Figure 7.7 shows the four information objects *Bill*, *Customer*, *Contract*, and *Order*. In the Best–Practice EA use case we decided to assign the values more fine grained than in the Living EA use case, i. e. for every schema entry a unique effort baseline is assigned. Table 7.10 shows the assignment. The effort baseline increases monotonous with the number of schema entries. The most complex information object is the *Contract* followed by the *Customer*. *Order* is considered to be less complex, as it has fewer schema entries. However, the information object *Bill* has the least schema entries.

	Information object schema entries										
Effort baseline	1	2	3	4	5	6	7	8	9	10	11
in person-month (PM)	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	2,0	2,1

Table 7.10: Information Object Schema Entries and Corresponding Effort Baseline in Person-Month PM for the Best–Practice EA - Use Case

We proceed with the assignment of values for the cost dimension in the Best–Practice EA use case. Table 7.11 shows the different values for fixed costs (C_f), cost of an application

(C_a), costs of an application service (C_s), and the variable costs (C_v). We assume the values to be taken from an IT service catalogue, if one is available in the enterprise. Otherwise, one has to estimate the values for costs.

Cost Value Dimension	Cost in Euros
C_f	3.000
C_a	15.000
C_s	1.750
C_v	3.500 $\times PM$

Table 7.11: Assignment of Costs to Cost Value Dimensions for the Best–Practice EA - Use Case

As, we were not able to ask an enterprise architect, regarding the scaling factors to be used and the adjustment of them in test scenarios, we decided to just slightly change the values used in the Living EA use case.

Scaling factor	Value
c_q	1,14
c_d	1,05
t_q	0,285
t_d	0,525

Table 7.12: Scaling Factors for Different Resource Modes in the Living EA - Use Case

7.3.2 Interactive Transformation Path Setting for the Best–Practice EA - Use Case

We now introduce applicable transformation actions for the Best–Practice EA use case and their ranking. Given the modeled situation in Figure 7.6 on the left side the two applications *Giga Z* and *WFT* are selectable for reuse in the order management process. Furthermore, it is possible to develop a new application in the order management process in the subsidiaries. For the customer management process three different applications are available for reuse and it is also possible to develop a new application.

We grouped the applicable transformation actions by their corresponding transformation action. In brackets behind the transformation action is the reference to the typed attributed graph production and the number of transitions in GG_{TD} created through it. The numbered TA's are used to reference the applicable transformation actions in the ranking that we introduce later on.

- Reuse application ($P_{ReuseApp}$ twenty-two):
  - $TA1$ Reuse *Giga Z* as *Order Management Application*
  - $TA2$ Reuse *WFT* as *Order Management Application*
  - $TA3$ Reuse *Elcaro CRM* as *Customer Relationship Management Application*
  - $TA4$ Reuse *PAS CRM* as *Customer Relationship Management Application*
  - $TA5$ Reuse *Total CRM* as *Customer Relationship Management Application*
- Develop new application (P_{NewApp} nine):
  - $TA6$ Develop a new application for *Order Management Application*
  - $TA7$ Develop a new application for *Customer Relationship Management Application*

Figure 7.8 shows the different states reachable through the application of transformation actions on S_{TD}. Overall, GG_{TD} creates fourty-one states and fifty-two transitions, whereas the twelve states at the bottom and the transitions to them can be neglected, as the application of $P_{intermediateGoalState_{TD}}$ adds no information to the graph, that is relevant for an enterprise architect.

Table 7.13 shows the ranking of the different transformation actions. As, the functional delta between *Elcaro CRM* and the other customer management applications is zero, reusing it results in the least effort. Furthermore, there is no difference between the minimum and maximum efforts, as no variable costs, time and risk are to be considered by this transformation action. The next best ranked transformation actions are $TA4$ and $TA5$ in the maximum frame that either reuse *PAS CRM* or *Total CRM*. In both cases the functional delta to *Elcaro CRM* has to be removed. As, the functional delta between both of them is zero, their efforts are equal and result in the same ranking. The same holds for the transformation actions in the minimum frame that are the next best ranked transformation actions.

They are followed by reusing *Giga Z* for order management. This transformation action results in less effort than reusing *WFT*, even though the functional delta between *Giga Z* and *WFT* vice versa is equal. This is a result of the information services and information objects involved in the functional delta that are to be realized by *Giga Z*. They differ regarding the schema entries of the information objects. Therefore, the next best ranked transformation action is $TA2$ in the maximum frame, that reuses *WFT*. $TA1$ and $TA2$ in the minimum frame are the next and after next best ranked transformations. They differ for the same reasons as their counterpart in the maximum frame. Next best ranked transformation action is the development of a new application for customer relationship management that needs to take care of the aggregated functional delta (see Definition

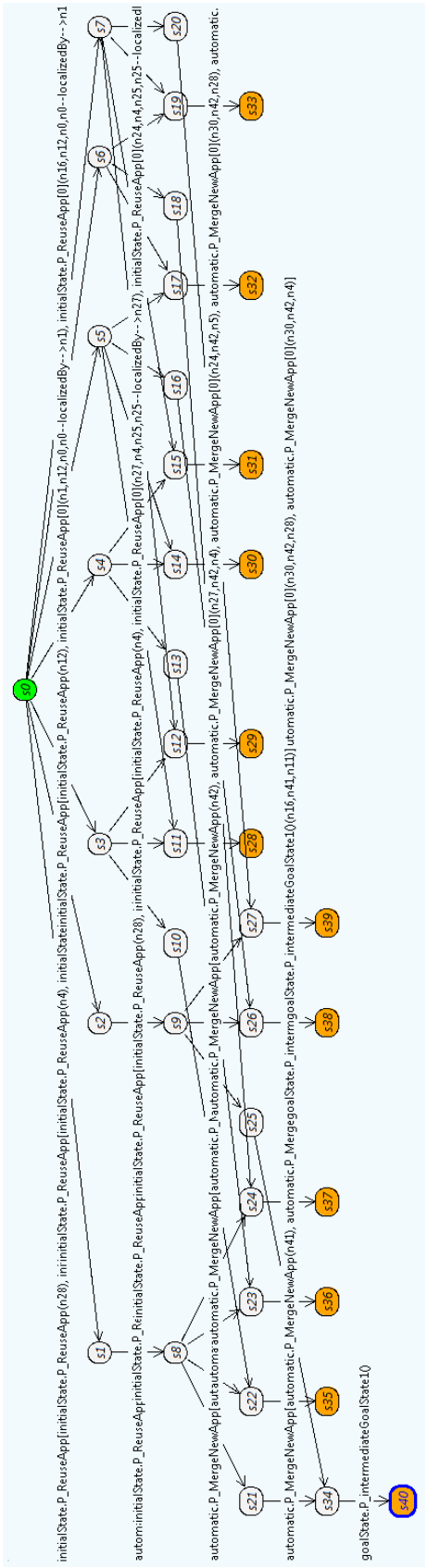

Figure 7.8: Possible Target Architectures Generated Through the Graph Grammar GG_{TD} in the First Phase for the Best–Practice EA - Use Case

29) of *PAS CRM*, *Total CRM*, and *Elcaro CRM*. Again the maximum frame is ranked better than the minimum frame. Second last ranked is the transformation action in the maximum frame for the development of a new application for order management, as it results in the second biggest effort. Last ranked is the same transformation action in the minimum frame.

Transformation Action	Min, Max Frame	Ranking Value
$TA3$	Min	22,84164510
$TA3$	Max	22,84164510
$TA4$	Max	65,45635243
$TA5$	Max	65,45635243
$TA4$	Min	66,22662503
$TA5$	Min	66,22662503
$TA1$	Max	104,10429710
$TA2$	Max	105,74994650
$TA1$	Min	110,67898540
$TA2$	Min	112,69107220
$TA7$	Max	143,53087360
$TA7$	Min	168,60993040
$TA6$	Max	221,36695610
$TA6$	Min	261,59571660

Table 7.13: Ranking Values with Minimum and Maximum in the Best–Practice EA - Use Case Ordered by Ranking Value

We decided to select $TA6$ in the maximum frame and $TA3$ that corresponds to the target business support map shown in Figure 7.5. After selecting $TA6$ the typed attributed graph production $P_{mergeNewApps}$ is selected automatically in the background to merge the nodes of type application, that are created through $TA6$. The resulting state generated through GG_{TD} is s35 from Figure 7.8. It is $G_{intermediate}$ and serves as the starting state for the second phase.

We do not describe the decisions of the second phase as they are similar to the decisions in the bottom-up approach.

7.3.3 Summary of the Best–Practice - Use Case

We demonstrated the top-down approach for selecting a target architecture with the Best–Practice EA use case. Given the business support maps we showed how the information is reused and allow for the derivation of applicable transformation actions. We provided an explanation on the framing mechanisms that allows for a decision support in

the first phase, without restricting the detailed decisions that follow in the second phase. Furthermore, we showed the setting of the different values and the resulting ranking for the different transformation actions applicable in the use case.

7.4 Development Master Data Management (DMDM) - Use Case

In the following we introduce the use case for the transformation path generation. Many enterprises developed in the past a variety of IT applications and established interconnections between them to address particular business needs of its departments. For the departments this way of transforming the IT is fine, as their business needs get satisfied in an appropriate time and manner. However, if we take a holistic and long-term view on the enterprise's IT it is not effective to store redundant data in different IT applications as it increases the risk of outdated and inconsistent data. Furthermore, the purely business driven addition of dependencies may lead in the long run to an increased replacement time of applications. This is the basis for the master data management challenge as introduced in Loshin (2009). In our use case we show an anonymized example for the introduction of the master data management approach in the research and development department of an organization. Therefore, we call it the development master data management (DMDM) use case.

7.4.1 Preliminaries for the DMDM - Use Case

Figure 7.9 shows the current architecture of the DMDM use case. An application to tackle the DMDM challenge was already introduced in the application landscape. It is called *DMDM system* and implements two application services *MasterData_v1* and *MasterData_v2*. However, not all existing applications have dependencies to the application services provided by the *DMDM system*. The *DevManager* provides similar data via the application service *QueryDev_v1*.

Two other applications use that application service: *Product planning tool* and the *Quality tests planning tool*. These two applications also use already the *MasterData_v2* application services. This application service is also used by two applications called *Physical quality test assistance tool* and *Physical quality test result database*. They allow for planning physical tests and saving the results of the physical tests. In contrast the application *Virtual quality test result database* provides this functionality for virtual testing results, for example created through simulations. It is not interconnected with other applications via application services. This means that currently the exchange of the results takes place manually.

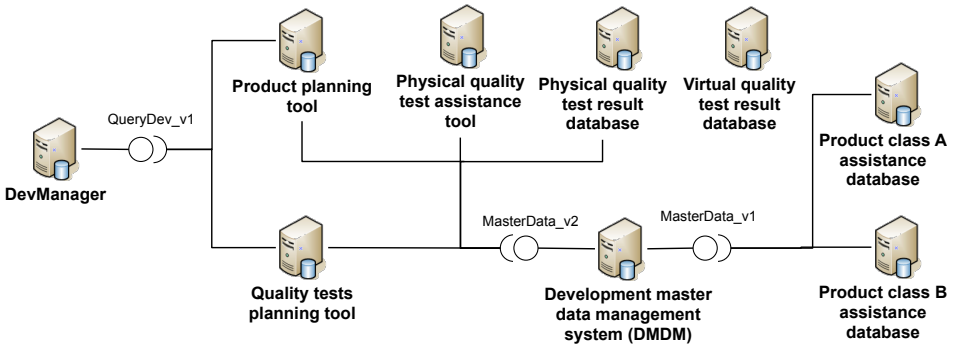

Figure 7.9: Current Architecture for the DMDM Use Case; Borrowed from Lautenbacher et al. (2013, Figure 4)

Furthermore, the current architecture contains two applications *Product class A assistance database* and *Product class B assistance database*, that allow to configure products for testing. Both differ in the product classes they allow to configure, not the type of tests (physical or virtual).

In the target architecture the functionality from different IT applications is going to be consolidated. Figure 7.10 shows the target architecture for the DMDM use case. The application services implemented by the *DMDM system* are consolidated. It provides the application service *MasterData_v3*, that will be used by the consolidated quality test assistance and result management tool and product modification assistance database applications. All quality tests and the results will be managed by one application: the quality test assistance and result management tool. There will be only one application for planning the product, as well as, the quality tests. This application is called *product and quality test planning tool*. It implements an application service *PlanningData_v1* that will be used by the quality test assistance and result management tool.

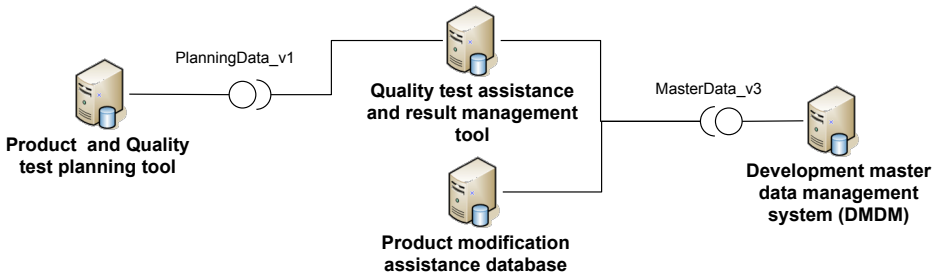

Figure 7.10: Target Architecture for the DMDM Use Case; Borrowed from Lautenbacher et al. (2013, Figure 5)

Figure 7.11 shows the transformation model of the DMDM use case. *Quality test assistance and result management tool* is a successor of the applications *Product planning tool* and *Quality tests planning tool*s. *Product class A assistance database* and *Product class*

B assistance database have as successor application the product modification assistance database. The *quality test assistance and result management database* is the successor application of the three applications: *Physical quality test assistance tool*, *Physical quality test result database*, and *Virtual quality test result database*. *DevManager* has no successor and the *DMDM system* is a successor of itself. The application service *QueryDev_v1* has no successor, whereas *PlanningData_v1* has no predecessor. Application service *MasterData_v3* is the successor of the application services with the lower version numbers.

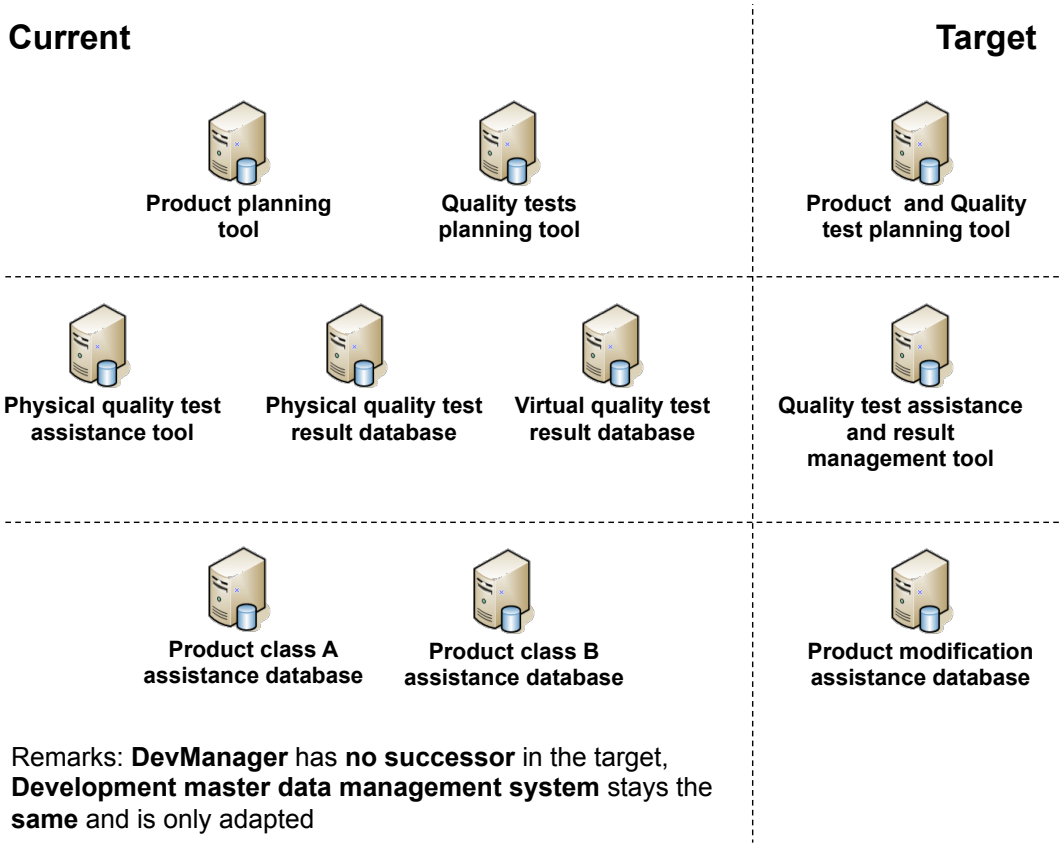

Figure 7.11: Transformation Model in the DMDM Use Case

Given the information of the current architecture, target architecture and transformation model of the DMDM use case we now proceed with the description of the initial state S_{TP} used for the generation of the transformation path. Figure 7.12 shows S_{TP}. It contains all applications and application services, from the current and target architecture, with their lifecycle phases. The successor edges are derived from the transformation model and all uses edges, that are present in the current architecture, are part of S_{TP}. Furthermore, the implements edges of the current and target architecture between applications and application services are part of S_{TP}. A goal node is also part of S_{TP}.

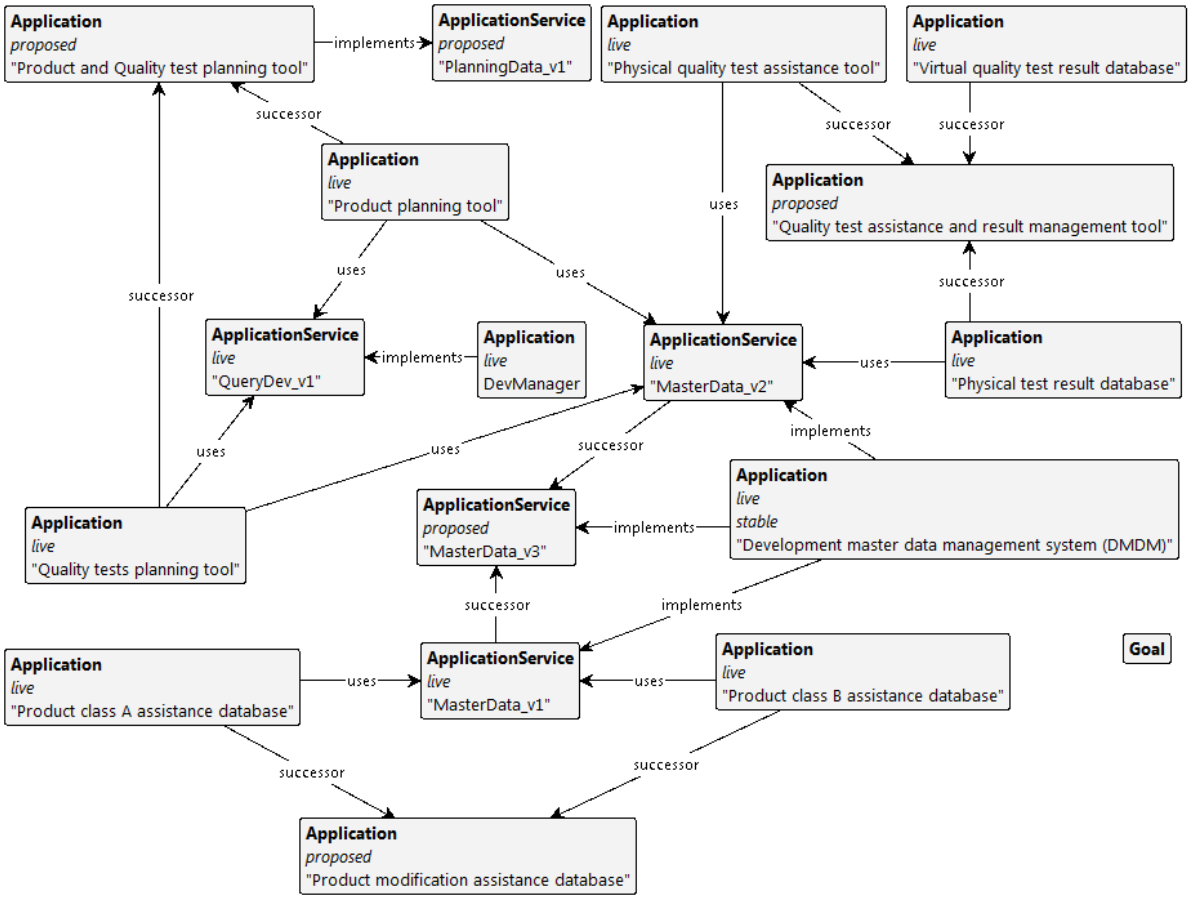

Figure 7.12: Initial State S_{TP} for the DMDM Use Case

The initial state S_{TP} is an excerpt of the overall graph that is independent plannable. Such an excerpt is called segment. Figure 7.13 shows two segments that are derivable, given the graphs of $EAG_{current}$, EAG_{target}, and $transformationModel$.

A segment is independent plannable because it contains an independent subgraph that has no dependencies to nodes of another subgraph that are changed. For details on how to create such segments we refer to Lautenbacher et al. (2013). Please note that the unchangedSuccessor edge corresponds to a successor edge between two nodes that belong to *stable*. The *DMDM system* of the use case is such a node. Creating such a segment has two advantages. Firstly, the state space and computation time for creating suggestions is reduced. Secondly, the enterprise architect is likely to plan transformation paths for areas of focus, that correspond to segments, and does not bustle from one part of the architecture to a randomly different part.

The goal state for the DMDM use case is detectable via a specialized typed attributed graph production $P_{goalState_{TP}}$. Figure 7.14 shows it. As soon as it is applicable the automated planner notifies the enterprise architect.

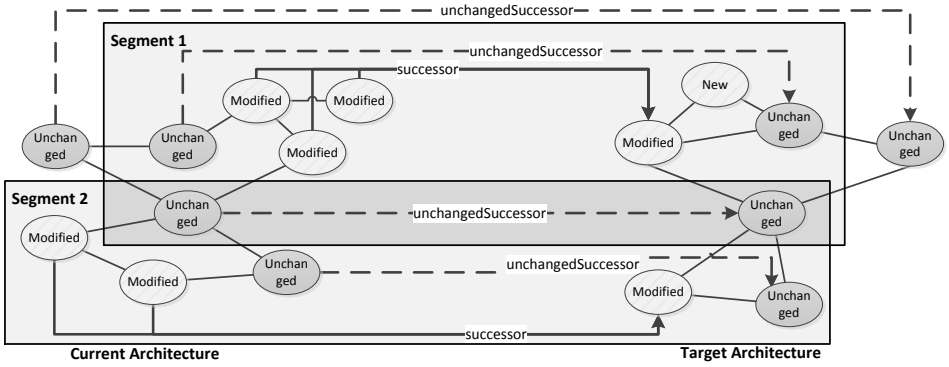

Figure 7.13: Segmentation of Elements in Current and Target Architecture to Derive Independent Plannable Subgraphs; Borrowed from Lautenbacher et al. (2013, Figure 2)

The application of $P_{goalState_{TP}}$ creates the reached attribute for the goal node that is already present in the initial state. It contains all applications and application services in their corresponding lifecycle phases in the target architecture. Furthermore, the usage and implementation dependencies of the target architecture are present. Figure 7.14 omits the implementation dependencies between the applications and application services that belong to *onlyCurrrent*. However, this is not an information necessary in the goal state, as these dependencies are not changed by transformation actions.

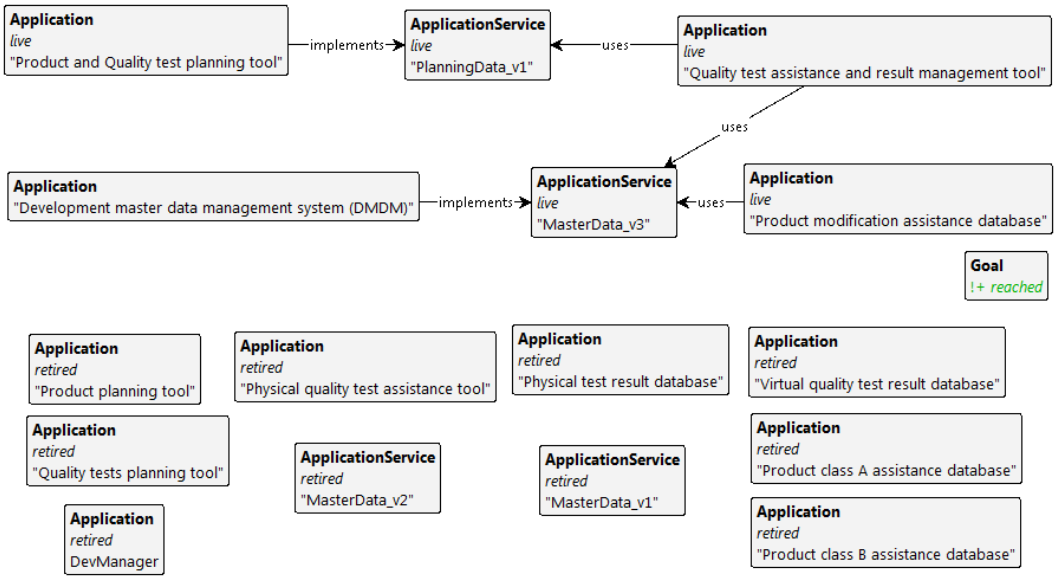

Figure 7.14: Goal State for the DMDM Use Case Formalized Via the Typed Attributed Graph Production $P_{goalState_{TP}}$

7.4.2 Interactive Generation of the Transformation Path for the DMDM - Use Case

In the following we describe the transformation path created by an enterprise architect for the DMDM use case. We start with giving an overview of applicable transformation actions and proceed with the assignment of them to steps and stages.

The available transformation actions are listed below, whereas the interdependencies between them are not considered and transformation actions to change dependencies are also neglected. The former cannot be described appropriately in the list way and the latter are dependent on the sequence of transformation actions. Every transformation action in the list below is sorted by the earliest step it can be assigned to. In brackets behind each transformation action is the reference to the typed attributed graph production.

- Step 1
  - Develop application ($P_{DevelopApp}$): *Product and quality test planning tool*
  - Develop application ($P_{DevelopApp}$): *Quality test assistance and result management tool*
  - Develop application ($P_{DevelopApp}$): *Product modification assistance database*
  - Develop application service ($P_{DevelopAS}$): *MasterData_v3*
- Step 2
  - Develop application service ($P_{DevelopAS}$): PlanningData_v1
- Step 3
  - Shutdown application ($P_{ShutdownApp_{succ}}$): *Product planning tool*
  - Shutdown application ($P_{ShutdownApp_{succ}}$): *Quality tests planning tool*
  - Shutdown all applications ($P_{ShutdownApp_{allSucc}}$): *Product planning tool* and *Quality tests planning tool*
  - Shutdown application service ($P_{ShutdownAS_{noSucc}}$): *QueryDev_v1*
  - Shutdown application service ($P_{ShutdownAS_{succ}}$): *MasterData_v1*
  - Shutdown application service ($P_{ShutdownAS_{succ}}$): *MasterData_v2*
  - Shutdown all application services ($P_{ShutdownAS_{allSucc}}$): *MasterData_v1* and *MasterData_v2*

- Step 4
  - Shutdown application ($P_{ShutdownApp_{noSucc}}$): *DevManager*
  - Shutdown application ($P_{ShutdownApp_{succ}}$): *Virtual quality test result database*
  - Shutdown application ($P_{ShutdownApp_{succ}}$): *Physical quality test assistance tool*
  - Shutdown application ($P_{ShutdownApp_{succ}}$): *Physical quality test result database*
  - Shutdown all applications ($P_{ShutdownApp_{allSucc}}$): *Virtual quality test result database*, *Physical quality test assistance tool*, and *Physical quality test result database*
  - Shutdown application ($P_{ShutdownApp_{succ}}$): *Product class A assistance database*
  - Shutdown application ($P_{ShutdownApp_{succ}}$): *Product class B assistance database*
  - Shutdown all applications ($P_{ShutdownApp_{allSucc}}$): *Product class A assistance database* and *Product class B assistance database*

With the applicable transformation actions at hand we proceed with describing the selection process. Figure 7.15 shows the result of the selection process of applicable transformation actions to the first three steps and two stages, for the DMDM use case.

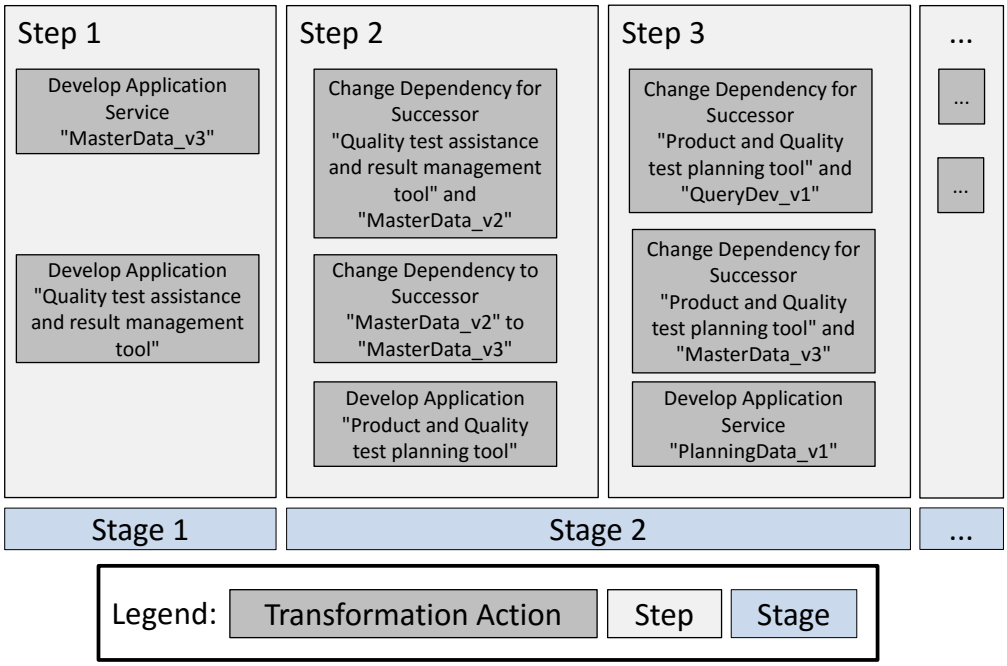

Figure 7.15: Assignment of Applicable Transformation Actions in the DMDM Use Case for Three Steps and Two Stages

The enterprise architect starts with selecting one of the transformation actions provided for the first step. She chooses to start with the development of the application service *MasterData_v3* and the development of the application *quality test assistance and result management tool*. Then step one is considered as finished even though it would be possible to develop two more applications. In step two she decides to change the dependencies of the applications that use *MasterData_v2* to *MasterData_v3*. The dependencies for the application created in step one are added and removed from its successors. Furthermore, she decided to develop the application Product modification assistance database. Even though, it would be already possible to develop *PlanningData_v1* it was not selected and step two was considered as finished. She proceeded with step three and selected to change the dependencies from the predecessors of *product and quality test planning tool* to this application for the application service *QueryDev_v1*. The same transformation action is applied for the dependencies to the application service *MasterData_v3*. Furthermore, she selected to developed the application service *PlanningData_v1*. Step three and stage two where considered as ready and she proceeded with the remaining applicable transformation actions. We do not describe the further selection process of transformation actions until $P_{goalState_{TP}}$ was applicable.

Figure 7.16 shows the state resulting from the application of the transformation actions shown in Figure 7.15.

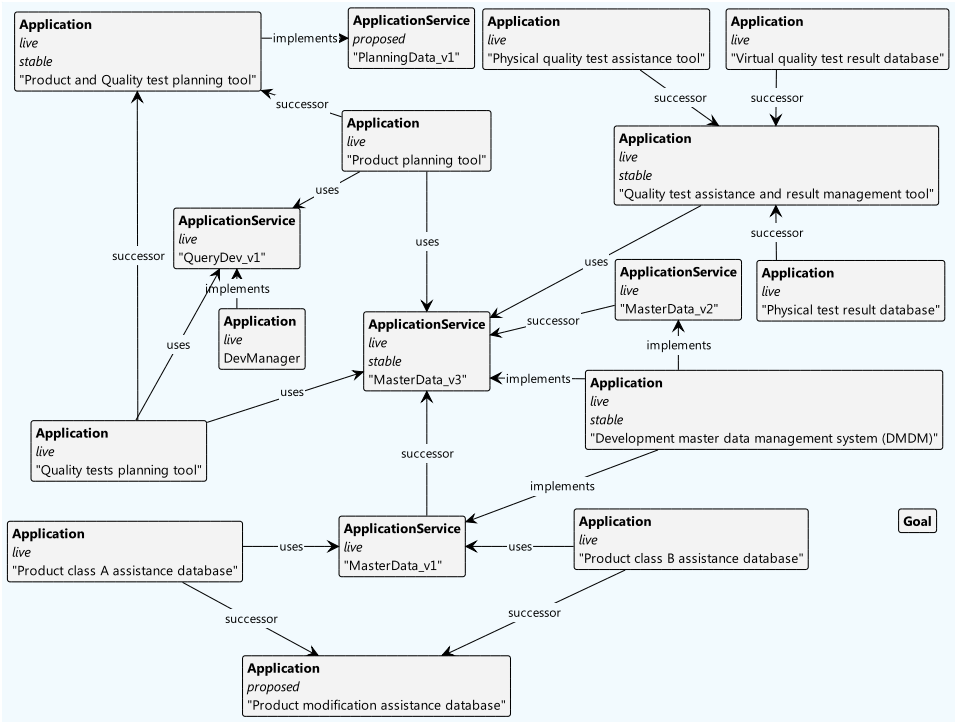

Figure 7.16: Intermediate State G_j after the Transformation Actions from the First Two Stages from Figure 7.15 are applied

The applications and application services that have already been selected to this point are already in the lifecycle phase live. Since, the shutdown of the predecessor applications and application services has not been selected yet, their lifecycle phases are also still live. The usage dependencies have already been removed from the application service *MasterData_v2* and the two applications *Quality tests planning tool* and *Product planning tool* do neither use any application services nor implement any. As a consequence it would be possible to shutdown *MasterData_v2* and the two applications in the next step. However, it would not be possible to shutdown *MasterData_v1* and *QueryDev_v1* as they are still used by applications. Another important aspect of the intermediate state is that the *product and quality test planning tool* uses *QueryDev_v1*. This was neither specified in the current nor the target architecture. However, the enterprise architect decided to create the dependency by selecting 'change dependency for successor: *Product planning tool*, *Quality tests planning tool* uses dependency to application service *QueryDev_v1*'. Removing the usage dependency requires a transformation action in $P_{TP_{Unique}}$ (see Subsubsection 5.1.2.3) to change unique dependencies, as the dependency is not part of EAG_{target}.

7.4.3 Summary of the DMDM - Use Case

We have shown how the transformation path for the DMDM use case was created and how the transformation actions were assigned to steps and stages. The graph grammar that is created in the background allows the enterprise architect to select the applicable transformation actions and hinders the creation of inconsistent states. Furthermore, the graph grammar hinders the generation of incorrect sequences of transformation actions. The created steps and stages, including the transformation actions, are the foundation to create a qualitative roadmap that serves for communicating and enacting the transformation in the enterprise.

7.5 Scenario-Based Evaluation

After evaluating our approach with different use cases we now proceed with a scenario-based evaluation. This type of evaluation is a valid method to evaluate software architectures (Zhu, 2005). As, we have not provided any deployment perspective and modular structure of our approach we use the scenarios to evaluate it on an implementation independent level. Nevertheless, we are able to evaluate the impact of each scenario on our approach and how we respond to it.

7.5.1 Modifiability of Transformation Actions

In the following we present scenarios that consider changes to the transformation actions that are selectable in interactive transformation path planning.

Scenario 1. *The enterprise architect wants to introduce new transformation actions or change existing with the elements present in the planning domain model.*

Impact of Scenario 1 on the Solution. The alternative transformation actions are influenced if a new alternative transformation action is added to the transformation action repository. Furthermore, the effects of the new transformation action might influence other transformation actions' preconditions. An adaptation of the typed attributed graph production to detect goal states might be necessary. However, the parts of the planning domain model relevant for the enterprise architect and are not impacted by this scenario.

Response to Scenario 1. First of all it is necessary to determine if the enterprise architect wants to consider further alternatives for existing transformation actions or if further transformation patterns are to be considered by transformation actions. In the former case it is necessary to gather all alternatives in question and decide whether further information needs to be included in the LHS, NAC, or RHS to allow for an integration of the new or adapted transformation action. In the latter case we can add the new transformation actions without considering alternatives. However, we need to consider if there are side effects of the new transformation actions with existing ones, that need to be considered from an enterprise architect's point of view. Technically the side effects are easily detectable, as we only need to check whether parts of transformation actions for different transformation patterns change information that is part of the other transformation actions NAC or LHS. Regardless, whether alternatives are added or new transformation actions for transformation patterns are added, it needs to be checked carefully if the goal states are impacted by the changes or addition.

Scenario 2. *The enterprise architect wants to consider transformation actions that use elements, including their relationships and attributes, currently not considered in the planning domain model.*

Impact of Scenario 2 on the Solution. This scenario implies that, besides new transformation actions, it is necessary to extend the planning domain model. An extension of the planning domain model does not impact the existing transformation actions. However, the extension might have an impact on the existing transformation actions by inducing an extension of these transformation actions as well. This might be the case if the enterprise architect needs additional information for decision making, currently not available, and wants to consider this information also in existing transformation actions. Obviously, this might impact the typed attributed graph productions used to detect goal

states, as the new elements and transformation actions using them result in extended models of the states.

Response to Scenario 2. After determining the transformation actions to be added it is possible to determine the necessary extensions to the planning domain model. An example would be a transformation action that allows to retire a technology component, like an application server, used by applications. Besides, checking side effects on existing transformation actions it is necessary to identify transformation patterns that are involved in the extension. We estimate that the impact of the side effects on existing transformation actions is low, as we classify this scenario rather being involved in a former or later decision phase. For example one would first decide about applications to be reused or developed and afterwards take technological decisions via transformation actions into account. Please note that this scenario considers the addition of transformation actions that change elements, relationships and attributes currently not available, without limiting the scope on the addition to ranking issues.

7.5.2 Modifiability of Ranking

Changes to the ranking of transformation actions are evaluated with the scenarios that follow.

Scenario 3. *The enterprise architect wants to consider additional or less factors in the ranking of transformation actions.*

Impact of Scenario 3 on the Solution. Removing factors from the ranking has no impact on our approach, but the lesser the diversity of the factors and their constituents the lesser the granularity of the ranking. Adding or removing factors has an impact on the weighting, as it needs to be reconfigured to consider the preferences of the enterprise architect. Additionally, it needs to be checked where the information from the new factors comes from. An extension of the planning domain model might be necessary. New transformation actions that are to be introduced trigger a change in the ranking or might even require a new composition of factors, as for example decisions regarding technological elements are based on different factors than in our approach. However, this scenario has no impact on the MCDM technique used.

Response to Scenario 3. The starting point should be the demand of the enterprise architect to adapt the factors involved in the ranking. In the case that factors are to be removed the weighting needs to be adapted, but no further adaptations are necessary. For the case where factors are added it is necessary to check whether elements need to be added to the planning domain model or if it is sufficient to add attributes and relationships. As an example an enterprise architect might want to consider the sustainability of the technical components that applications run on. One might add those technical

components and their relationships to the planning domain model to consider them in the ranking. However, we would suggest to aggregate the sustainability of the underlying technical components into an attribute of the applications that can be considered in the ranking. Such an aggregated information for applications may be called 'Application Fitness Index' (c. f. Durst (2008)). We advise to use such aggregated information as far as possible, i. e. if the information is not changed by transformation actions and the aggregation does not result in a loss of information relevant for the automated planner. Then the information should be aggregated into an attribute of an existing element.

Scenario 4. *The enterprise architect wants to consider qualitative factors in an MCDM technqiue used for ranking transformation actions.*

Impact of Scenario 4 on the Solution. Even though the enterprise architect should not be involved in the decision regarding the MCDM technique to be used, she restricts the possible techniques to be used by postulating factors to be considered and their types. This scenario impacts the MCDM technique in use and requires the usage of a technique that allows the consideration of qualitative factors and their weighting. The planning domain model is also impacted by this scenario, as the information on the qualities needs to be integrated into the models to allow a consideration in the ranking.

Response to Scenario 4. As a MCDM technique that allows to consider qualitative factors one could use the Analytic Hierarchy Process introduced in Subsection 2.6.2. However, it is necessary to point out that using such techniques requires a complete enumeration of all alternatives that are to be ranked and then apply pairwise comparisons. We suggest to limit the number of possible alternatives a priori to reduce the number of comparisons involved in the ranking. Furthermore, it is important to use a dimensionless MCDM technique if different dimensions are used within the factors to calculate the ranking. In general the MCDM technique used to calculate the ranking can be exchanged, but it is necessary to consider advantages and drawbacks of the technique as well as computational issues regarding the number of transformation actions that are to be ranked.

Scenario 5. *The enterprise architect wants to consider different resource modes for transformation actions or differentiate between resources.*

Impact of Scenario 5 on the Solution. The impact of adding additional resource modes to our approach is minor, as long as the number of possible alternatives stays countable and the resource modes have discrete values. Adding different types of resources has a bigger impact on our approach, as we currently only consider efforts based on functionality and resulting person-month. If several other resources are to be added more sophisticated means are necessary to compute non-applicable transformation actions due to resource constraints. This scenario, if demanded by an enterprise architect, makes the alternatives to be computed a scheduling problem, which is out of scope of the thesis.

Response to Scenario 5. Adapting the resource consumption of transformation actions is adaptable easily if the scaling factors of time and costs are defined by a function that allows to determine their impact on the respective dimension. For example if we know the effort in normal resource mode, i.e. standard resource mode, we could derive the effort for a half or quintuple resource mode. Regarding the addition of different types of resources we refer to the proposed hybrid planning and scheduling mechanisms by Schattenberg (2009). Nevertheless, we suggest to reuse the transformation actions for exploring the possible solutions and to integrate them into the hierarchical task networks used in hybrid planning approaches.

Scenario 6. *The enterprise architect wants to consider benefits besides efforts and integrate both into an utility function.*

Impact of Scenario 6 on the Solution. For this scenario we assume that the calculation of the efforts remains as proposed by our approach. The introduction of benefits needs to be derivable from the models and as a consequence the planning domain model needs to be extended. Furthermore, it is necessary to determine how the benefits raise or lower through different transformation actions and their resource modes.

Response to Scenario 6. Using an utility function as the addition of benefits and negative efforts could be integrated by including attributes in the planning domain model that allow to determine the benefit of a transformation action. By determining further aspects for changes, like urgency and importance, it would be even possible to assign benefit values to the different resource modes of the transformation actions. Furthermore, one could restrict the selectable transformation actions to those that have non-negative utility. For using an utility function for combining availability (benefit) and costs (effort) based on continuous functions, in the context of enterprise architectures, we refer to Österlind et al. (2013). The benefit could also be ranked using the Weighted Product Model, like in the ranking of the efforts and the utility is then calculated as a sum of positive benefits and negative efforts.

7.5.3 Modifiability of Metamodels and Models

The scenarios we present in the following are concerned with challenges that may arise due to differences in metamodels or information in the available models.

Scenario 7. *The enterprise architect uses a different metamodel than the planning domain model.*

Impact of Scenario 7 on the Solution. There are three different cases that we need to consider for this scenario. Firstly, the metamodel in question includes all the

concepts of the planning domain model. Secondly, the planning domain model includes the metamodel in question completely. In both cases our approach is not impacted, as we have the same expressiveness of the metamodels and naming issues can be resolved through labeling mechanisms. Such labeling is a established method in OWL ontologies and allows to hide the different naming of concepts in the planning domain model from an enterprise architect. For example if the metamodel of the enterprise architecture uses the term 'application component' instead of the 'application' used in the planning domain model or the relationship between applications and application services should be named 'provides' and 'consumes' instead of 'implements' and 'uses', it is possible to solve these issues through labeling. Within the type graph such differences could also be integrated through including different attributes for the used namespaces.

Thirdly, and probably the most common case, is when only a partial overlapping between the planning domain model and metamodel in question exists. In this case the metamodel in question needs to be extended. Please note that we do not consider the equality of the planning domain model with the metamodel in question, as well as the case in which there is no commonality between them. The former case is ignored as the scenario is concerned with differences and the latter case is considered as handable with minor efforts as we have shown the interconnectedness of the planning domain model to widely accepted and established metamodels in Section 3.6.

Response to Scenario 7. We argue that only the concepts present in the planning domain model need to be integrated into the metamodel in question, to allow for a reuse of the transformation actions and their ranking. The necessary information needs to be gathered and modeled and allows an integration into the planning domain model. If the method used for planning transformations is not combinable with business support maps we suggest to model new transformation actions and adapt the planning domain model accordingly. If the metamodel and the planning domain model only intersect partially, we would prefer to add information as far as possible automatically. How this is done we have shown in the Best–Practice EA use case, where information services are missing. However, there maybe cases in which such an automatic addition might be not possible, if for example the metamodel in question does not consider business support elements it is not possible to derive them automatically. This is not possible because the business support elements are reified ternary relationships that cannot be reconstructed from binary relationships.

Scenario 8. *The enterprise architect models incomplete information regarding the current business support maps and current architecture.*

Impact of Scenario 8 on the Solution. If the modeled information is incomplete our approach does not create incorrect results in this scenario, as we did not provide transformation actions for transformation pattern (a) in Figure 4.3. If we specify that a transformation action is applicable to transformation pattern (a) it would also become applicable to an incompletely modeled transformation pattern (b) or (c) in Figure 4.3.

Furthermore, it might not be possible to compute applicable transformation actions and as a consequence transformation paths.

Response to Scenario 8. We suggest to check the completeness of the models a priori as part of a quality assurance process in the creation and maintenance of the models. Even though typed attributed graph productions provide the means to check for such incompleteness in the models (see debugging actions in Subsection 7.1.3) we recommend to check these issues in the enterprise architecture tool a priori. In the semantic web technology stack this task is supported through SPARQL queries that allow to receive for example all application services that do not realize an information service or all information services that are not realized by an application service. Using the automated planner in the background to automatically create a transformation path with a depth-first search allows to provide a fast feedback to the enterprise architect if it is possible to create transformation paths. If not the quality assurance process for the models should be triggered. Referring to the misinterpretation of modeled transformation patterns we suggest to always notify the enterprise architect that a change in the current business support map at hand has an impact on corresponding target business support maps.

Scenario 9. *The enterprise architect models incomplete information regarding the target business support maps and target architecture.*

Impact of Scenario 9 on the Solution. If the modeled information is incomplete our approach does not create incorrect results in this scenario. However, it might not be possible to compute transformation paths as expected from the enterprise architect. For example the typed attributed graph production $P_{goalState_{TP}}$, used in the transformation path generation to detect goal states and bound to individuals, might be reachable with less transformation actions. Furthermore, if transformation actions have been specified for transformation patterns, that may be misinterpreted through incomplete information in the models, as different transformation patterns, our approach would be impacted. For example transformation actions specified for transformation pattern (b) or (c) in Figure 4.3 might get misinterpreted as transformation pattern (f) in Figure 4.3.

Response to Scenario 9. A fast feedback can be provided to the enterprise architect if it is possible to create a transformation path, by using the automated planner in the background to automatically create a transformation path with a depth-first search. If not the quality assurance process for the models should be triggered, like in the case of incomplete models of the current business support maps and current architecture. Furthermore, the enterprise architect should always start modeling target business support maps and target architecture with a copy of the current counterparts. This ensures that nothing is forgotten accidentally. Of course the enterprise architect should be able to fade out information that is irrelevant for her current tasks. If changes at the currently edited model have an impact on other existing models she should be notified what the impact is in terms of the changing transformation patterns.

We consider this tasks to be supported by an appropriate query language. In our approach SPARQL queries support these tasks.

Scenario 10. *The enterprise architect has modeled an incomplete transformation model manually.*

Impact of Scenario 10 on the Solution. The decision support for selecting a target architecture (see Chapter 4) is not influenced by this scenario as in this phase the transformation model gets created automatically. However, the transformation path generation is impacted by an incomplete transformation model. Using an incomplete transformation model might result in incorrect sequences of transformation actions.

Response to Scenario 10. We advise to use the interactive approach suggested in Chapter 4 to avoid incomplete transformation models. Nevertheless, manual changes to the transformation model might be modeled through an enterprise architect. Therefore, an incomplete transformation model might be the result. Using the gap analysis allows to identify all individuals that belong to *stable* and as a consequence all individuals that are successor of themselves. For the used mechanisms we refer to Diefenthaler and Bauer (2013). For the cases where successor relationships between different individuals have been removed, more sophisticated means are necessary to reconstruct the transformation model. Such a sophisticated means is a similarity analysis of nodes in the current and target architecture that allows to suggest possible successors of certain individuals. For details regarding the analysis we refer to Lautenbacher et al. (2013, p. 58).

Scenario 11. *The enterprise architect wants to determine and extract explicitly intermediate states in the transformation path generation.*

Impact of Scenario 11 on the Solution. This scenario has no direct impact on our approach, as the used graph formalisms allow to determine the states resulting from the application of transformation actions. Furthermore, it is possible to determine the sequence of transformation actions that lead to a certain state. Our approach would be impacted if certain intermediate states should be marked like the goal states.

Response to Scenario 11. Referring to the impact on our approach to mark certain intermediate states we would have to introduce typed attributed graph productions that allow us to mark these states. This could be done automatically, like for the goal states, or a manual selection by the enterprise architect could be realized. As a consequence the enterprise architect could mark states to be extracted and reused later on and proceed with exploring further transformation actions and resulting states.

8 Summary

We end the thesis with providing a summary of the obtained results, future work and conclusions. Research objective number four of the thesis was already discussed and summarized in Chapter 7. Therefore, we will focus within the obtained results section on the remaining three objectives of the thesis. Future work addresses issues that we could not consider within the scope of the thesis, but is worth to be considered in future research efforts. The conclusions will round up the thesis with final notes on the insights of the thesis at hand.

8.1 Obtained results

In the following we will refer to the objectives of the thesis and summarize the obtained results presented in the main chapters of the thesis.

Research objective number one was concerned with the formalization of the concepts involved in transformation planning. We provided an overview of our conception of the concepts involved and used transformation patterns to formalize them. Furthermore, we presented requirements for a planning domain model that we formalized via a type graph. The objective was achieved and allowed us to build on these results in the subsequent chapters.

The formalization of the concepts, especially the set theoretical aspects, were important to establish a common terminology within the thesis that in turn enables a proper interaction of the enterprise architect with the solution later on. Furthermore, it was important to show that the transformation patterns and the planning domain model are largely already considered in widely adopted and mature EA metamodels.

The second research objective of the thesis was achieved by using business support maps and transformation actions as a means to select a target architecture interactively. We formalized the transformation actions via typed attributed graph productions that allow for an automatic derivation of applicable changes. Furthermore, we provided a ranking for the different transformation actions to give the enterprise architect an overview of the efforts resulting from them.

However, the enterprise architect has always the control regarding the decisions to be selected and the target architecture that results from these selections. To be able to reflect the preferences an enterprise architect has regarding different dimensions involved in the ranking of transformation actions we used a dimensionless MCDM technique with weights.

These results were important, as they showed that it is possible to allow for an interactive selection of a target architecture and the generation of the resulting target architecture. Furthermore, the ranking and the employment of different resource modes allowed to further differentiate between resulting states within the path generation and the weights of the state transitions. Last but not least it was shown that it is possible to determine a target architecture in the bottom-up as well as the top-down approach.

The third research objective of the thesis was concerned with enabling an interactive transformation path generation that considers the actual sequences of changes. This objective was achieved by considering different transformation patterns and transformation actions. By decoupling the interactive decision support for a target architecture from the transformation path generation, we allow for different approaches than ours introduced in Chapter 4, to determine a target architecture. This ensures the reusability of our approach.

8.2 Future Work

In this section we discuss future work that could build on the results of the thesis at hand. Given our ranking for transformation actions in the solution we do not consider yet the side effects on the ranking of the selection of a transformation action at a former or later point in the decision making process. This would lead to changes in the ranking of all remaining transformation actions after another has been selected. We assume that such side effects exist in reality, but we were not able to determine them with the enterprise architects and thus could not formalize them, at least to a certain degree. With such information at hand the decision space would have to be explored with more sophisticated means. Common search heuristics like A* (Russell and Norvig, 2010, Chapter 5) for partially explored decision spaces or the network simplex algorithm (Jungnickel, 2013, Chapter 11) for completely explored decisions spaces seem to be suitable to find optimal solutions for these cases and could aid the enterprise architect.

Future work should also address a combination of top-down and bottom-up planning like suggested by Aier et al. (2011). Our approach provides just a one way planning support but not a simultaneously planning in both directions. Such a planning support would require more sophisticated means that allow to consider the interplay of top-down decisions on bottom-up decisions and vice versa.

Lastly, we would like to extend our approach to be able to support in technology related decisions. Hanschke (2013) provides a methodology that allows to make technology related decisions based on landscape maps, that are the general type of visualization of business support maps that we used for enabling decision support in target architecture selection in Chapter 4. Therefore, we could support those decisions with the same mechanisms as described in this thesis, by just modeling transformation actions considering elements and relationships from a technology architecture and adapt the ranking in an appropriate way. This extension seems to be a promising development of the approach presented within this thesis.

8.3 Conclusions

The thesis at hand provides an automated planning view on the topic of transformation path planning. However, in contrast to classical automated planning approaches we decided to develop an interactive approach that allows the enterprise architect to become part of the plan generation. To be able to support the enterprise architect to estimate the value of a certain change we provided a ranking that the enterprise architect can adjust to her preferences. By providing a formalization of possible changes through transformation actions we allow to shift the focus from a model oriented perspective to a decision and model oriented one. Furthermore, using transformation patterns allows to abstract from situations to common patterns that are reused in transformation actions to ensure their applicability in different contexts.

Bibliography

Agievich, V. and Skripkin, K. (2014). Enterprise Architecture Migration Planning Using the Matrix of Change. *Procedia Computer Science*, 31:231–235.

Aier, S., Buckl, S., Gleichauf, B., Matthes, F., Schweda, C. M., and Winter, R. (2011). Towards a More Integrated EA Planning: Linking Transformation Planning with Evolutionary Change. In Nüttgens, M., Thomas, O., and Weber, B., editors, *Enterprise modelling and information systems architectures*, volume P-190 of *Lecture Notes in Informatics (LNI) - Proceedings*, pages 23–36. Gesellschaft für Informatik, Bonn.

Aier, S. and Gleichauf, B. (2010a). Application of Enterprise Models for Engineering Enterprise Transformation. *Enterprise Modelling and Information Systems Architectures*, 5(1):56–72.

Aier, S. and Gleichauf, B. (2010b). Towards a Systematic Approach for Capturing Dynamic Transformation in Enterprise Models. In Sprague, R. H., editor, *Proceedings of the 43rd Annual Hawaii International Conference on System Sciences*, pages 1–10. IEEE Computer Society, Los Alamitos.

Aier, S., Gleichauf, B., Saat, J., and Winter, R. (2009). Complexity Levels of Representing Dynamics in EA Planning. In Albani, A., Barjis, J., and Dietz, J. L. G., editors, *Advances in Enterprise Engineering III*, pages 55–69. Springer, Heidelberg.

Aier, S. and Winter, R. (2009). Virtual Decoupling for IT/Business Alignment – Conceptual Foundations, Architecture Design and Implementation Example. *Business & Information Systems Engineering*, 1(2):150–163.

Alexander, C., Ishikawa, S., and Silverstein, M. (1977). *A pattern language: Towns, buildings, construction*, volume 2 of *Center for Environmental Structure series*. Oxford University Press, New York.

Allemang, D. and Hendler, J. A. (2011). *Semantic Web for the working ontologist: Effective modeling in RDFS and OWL*. Morgan Kaufmann, Waltham, MA, 2 edition.

Andrikopoulos, V., Gmez Sez, S., Leymann, F., and Wettinger, J. (2014). Optimal Distribution of Applications in the Cloud. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Kobsa, A., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Terzopoulos, D., Tygar, D., Weikum, G., Jarke, M.,

Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., and Horkoff, J., editors, *Advanced Information Systems Engineering*, volume 8484 of *Lecture Notes in Computer Science*, pages 75–90. Springer International Publishing, Cham.

Binz, T., Fehling, C., Leymann, F., Nowak, A., and Schumm, D. (2012a). Formalizing the Cloud through Enterprise Topology Graphs. In *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 742–749.

Binz, T., Leymann, F., Nowak, A., and Schumm, D. (2012b). Improving the Manageability of Enterprise Topologies Through Segmentation, Graph Transformation, and Analysis Strategies. In *2012 16th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2012)*, pages 61–70.

Birkmeier, D. Q., Gehlert, A., Overhage, S., and Schlauderer, S. (2013). Alignment of Business and IT Architectures in the German Federal Government: A Systematic Method to Identify Services from Business Processes. In *46th Hawaii International Conference on System Sciences (HICSS)*, pages 3848–3857.

Boneva, I., Hermann, F., Kastenberg, H., and Rensink, A. (2007). Simulating Multigraph Transformations Using Simple Graphs. In Ehrig, K., Giese, H., Margaria, T., Padberg, J., and Taentzer, G., editors, *Proceedings of the Sixth InternationalWorkshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007)*.

Brandt, C. (2013). *An Enterprise Modeling Framework for Banks Using Algebraic Graph Transformation*. PhD thesis, Technische Universität Berlin.

Brealey, R. A., Myers, S. C., and Allen, F. (2008). *Principles of corporate finance*. The McGraw-Hill/Irwin series in finance, insurance, and real estate. McGraw-Hill/Irwin, Boston, 9th ed edition.

Bucher, T., Fischer, R., Kurpjuweit, S., and Winter, R. (2006). Analysis and Application Scenarios of Enterprise Architecture: An Exploratory Study. In *10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW'06)*, page 28. IEEE.

Buckl, S., Ernst, A. M., Lankes, J., and Matthes, F. (2008). *Enterprise Architecture Management Pattern Catalog: Version 1.0*. Technische Universität München, Munich, Germany.

Buckl, S., Ernst, A. M., Matthes, F., and Schweda, C. M. (2009a). An Information Model Capturing the Managed Evolution of Application Landscapes. *Journal of Enterprise Architecture*, 5(1):12–26.

Buckl, S., Ernst, A. M., Matthes, F., and Schweda, C. M. (2009b). Constructing an Information Model for Application Landscape Management. In *21st International Conference on Advanced Information Systems (CAiSE'09)*, pages 55–60.

Buckl, S., Matthes, F., and Schweda, C. M. (2011). Modeling Enterprise Architecture Transformations. In *The International IFIP WG5.8 Working Conference on Enterprise Interoperability (IWEI 2011)*.

Buckl, S. and Schweda, C. M. (2011). *On the State-of-the-Art in Enterprise Architecture Management Literature*. Technische Universität München.

Chen, W., Hess, C., Langermeier, M., Stülpnagel, J. v., and Diefenthaler, P. (2013). Semantic Enterprise Architecture Management. In *15th International Conference on Enterprise Information Systems (ICEIS 2013)*, pages 318–325.

Clark, T., Barn, B. S., and Oussena, S. (2011). LEAP: A Precise Lightweight Framework for Enterprise Architecture. In Bahulkar, A., Kesavasamy, K., Prabhakar, T. V., and Shroff, G., editors, *Proceedings of the 4th India Software Engineering Conference*, pages 85–94.

Dam, H. K., Le, L.-S., and Ghose, A. (2010). Supporting Change Propagation in the Evolution of Enterprise Architectures. In *2010 14th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2010)*, pages 24–33.

Dam, H. K. and Winikoff, M. (2008). Cost-based BDI plan selection for change propagation. In Padgham, L., Parkes, D., Müller, J., and Parsons, S., editors, *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems (AAMAS 2008)*, pages 217–224.

Dern, G. (2009). *Management von IT-Architekturen: Leitlinien für die Ausrichtung, Planung und Gestaltung von Informationssystemen*. Praxis. Vieweg + Teubner, Wiesbaden, 3. edition.

Diefenthaler, P. and Bauer, B. (2013). Gap Analysis in Enterprise Architecture using Semantic Web Technologies. In *15th International Conference on Enterprise Information Systems (ICEIS 2013)*, pages 211–220.

Diefenthaler, P. and Bauer, B. (2014a). From Gaps to Transformation Paths in Enterprise Architecture Planning. In Hammoudi, S., Cordeiro, J., Maciaszek, L. A., and Filipe, J., editors, *Enterprise Information Systems*, volume 190 of *Lecture Notes in Business Information Processing*, pages 474–489. Springer International Publishing, Cham.

Diefenthaler, P. and Bauer, B. (2014b). Using Gap Analysis to Support Feedback Loops for Enterprise Architecture Management. In Kundisch, D., Suhl, L., and Beckmann, L., editors, *MKWI 2014 - Multikonferenz Wirtschaftsinformatik*, Paderborn.

Diefenthaler, P., Langermeier, M., and Bauer, B. (2015). Interactive Transformation Path Generation Using Business Support Maps. In Barjis, J. and Pergl, R., editors, *Enterprise and Organizational Modeling and Simulation, 11th International Workshop, EOMAS 2015, Held at CAiSE 2015*, Lecture Notes in Business Information Processing, Heidelberg. Springer.

Durst, M. (2008). *Wertorientiertes Management von IT-Architekturen*. Wirtschaftsinformatik. Deutscher Universitäts-Verlag / GWV Fachverlage GmbH, Wiesbaden, Wiesbaden.

Edelkamp, S. and Hoffmann, J. (2004). *PDDL2.2: The Language for the Classical Part of the 4th International planning Competition*, volume Technical Report No. 195. Institut für Informatik.

Edelkamp, S. and Rensink, A. (2007). Graph Transformation and AI Planning. In Edelkamp, S. and Frank, J., editors, *Knowledge Engineering Competition (ICKEPS)*, Rhode Island, USA.

Ehrig, H., Ehrig, K., Prange, U., and Taentzer, G. (2006). *Fundamentals of algebraic graph transformation*. Monographs in theoretical computer science. Springer, Berlin and New York.

Ehrig, H., Heckel, R., Korff, M., Löwe, M., Ribeiro, L., Wagner, A., and Corradini, A. (1997). Algebraic Approaches to Graph Transformation - Part II: Single Pushout Approach and Comparison with Double Pushout Approach. In Rozenberg, G., editor, *Handbook of graph grammars and computing by graph transformation*, volume 1, pages 247–312. World Scientific, River Edge, NJ, USA.

Engels, G., Hess, A., Humm, B., Juwig, O., Lohmann, M., and Richter, J.-P. (2008). *Quasar Enterprise: Anwendungslandschaften serviceorientiert gestalten*. dpunkt.verlag, Heidelberg, 1 edition.

Ernst, A. M. (2010). *A Pattern-based Approach to Enterprise Architecture Management*. PhD thesis, Technische Universität München, München.

Estler, H.-C. and Wehrheim, H. (2011). Heuristic Search-Based Planning for Graph Transformation Systems. In Bartak, R., Fratini, S., McCluskey, L., and Vaquero, T. S., editors, *KEPS 2011 Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling*, pages 54–61.

Fishburn, P. C. (1967). Additive Utilities with Incomplete Product Sets: Application to Priorities and Assignments. *Operations Research*, 15(3):537–542.

Fowler, M. (2001). *Analysis patterns: Reusable object models*. Addison-Wesley, Boston, 10 edition.

Franke, U., Holschke, O., Buschle, M., Narman, P., and Rake-Revelant, J. (2010). IT Consolidation: An Optimization Approach. In *2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW)*, pages 21–26.

French, S., Maule, J., and Papamichail, N. (2009). *Decision behaviour, analysis and support*. Cambridge University Press, Cambridge, United Kingdom.

Gamma, E. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley professional computing series. Addison-Wesley, Reading, Mass.

Gerber, A., Kotzè, P., and van der Merwe, A. (2010). Towards the Formalisation of the TOGAF Content Metamodel using Ontologies. In Filipe, J. and Cordeiro, J., editors, *ICEIS 2010*, volume 2, pages 54–64. SciTePress.

Ghallab, M., Nau, D. S., and Traverso, P. (2004). *Automated planning: Theory & practice*. Morgan Kaufmann and Elsevier Science, San Francisco and Calif and Oxford.

Ghamarian, A. H., Mol, M., Rensink, A., Zambon, E., and Zimakova, M. (2012). Modelling and analysis using GROOVE. *International Journal on Software Tools for Technology Transfer*, 14(1):15–40.

Gleichauf, B. (2011). *Planung der Unternehmensarchitektur: Beiträge zur Entwicklung einer Methode für die Planung der Transformation von Unternehmensarchitekturen auf Basis von Unternehmensarchitekturmodellen und unter Berücksichtigung dynamischer Aspekte*. PhD thesis, University of St. Gallen.

Glissmann, S. M. and Sanz, J. (2011). An Approach to Building Effective Enterprise Architectures. In *2011 44th Hawaii International Conference on System Sciences (HICSS 2011)*, pages 1–10.

Goethals, F. G., Snoeck, M., Lemahieu, W., and Vandenbulcke, J. (2006). Management and enterprise architecture click: The FAD(E)E framework. *Information Systems Frontiers*, 8(2):67–79.

Goetz, T. and Maurer, P. (2011). Musterbasierte Anwendungsintegration. In Strahringer, S., editor, *Application Management*, volume 278 of *HMD Praxis der Wirtschaftsinformatik*, pages 78–87. dpunkt.verlag, Heidelberg.

Greefhorst, D. and Proper, E. (2011). *Architecture Principles*. The Enterprise Engineering Series. Springer Berlin Heidelberg, Berlin and Heidelberg.

Grimm, S., Watzke, M., Hubauer, T., and Cescolini, F. (2012). Embedded EL + Reasoning on Programmable Logic Controllers. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J. X., Hendler, J., Schreiber, G., Bernstein, A., and Blomqvist, E., editors, *The Semantic Web – ISWC 2012*, Lecture Notes in Computer Science, pages 66–81. Springer Berlin Heidelberg, Berlin and Heidelberg.

Gringel, P. and Postina, M. (2010). I-Pattern for Gap Analysis. In Engels, G., Luckey, M., Pretschner, A., and Reussner, R., editors, *Software engineering 2010*, Lecture Notes in Informatics, pages 281–292. Gesellschaft für Informatik, Bonn.

Gutzwiller, T. A. (1994). *Das CC RIM-Referenzmodell für den Entwurf von betrieblichen, transaktionsorientierten Informationssystemen.* Betriebs- und Wirtschaftsinformatik. Physica-Verlag, Heidelberg.

Habel, A., Heckel, R., and Taentzer, G. (1996). Graph Grammars with Negative Application Conditions. *Fundamenta Informaticae*, 26(3,4):287–313.

Habel, A. and Pennemann, K.-H. (2005). Nested Constraints and Application Conditions for High-Level Structures. In Kreowski, H.-J., Montanari, U., Orejas, F., Rozenberg, G., and Taentzer, G., editors, *Formal Methods in Software and Systems Modeling*, volume 3393 of *Lecture Notes in Computer Science*, pages 293–308. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hanschke, I. (2009). *Strategisches Management der IT-Landschaft: Ein praktischer Leitfaden für das Enterprise Architecture Management.* Hanser, München, 1. edition.

Hanschke, I. (2012). C Planungs-Muster: Download-Anhang zum Buch Strategisches Management der IT-Landschaft.

Hanschke, I. (2013). *Strategisches Management der IT-Landschaft: Ein praktischer Leitfaden für das Enterprise-Architecture-Management.* Hanser, München, 3 edition.

Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design Science in Information Systems Research. *MIS Quaterly.*

Holt, R. C., Schürr, A., Sim, S. E., and Winter, A. (2006). GXL: A graph-based standard exchange format for reengineering. *Science of Computer Programming*, 60(2):149–170.

Iacob, M.-E., Quartel, D., and Jonkers, H. (2012). Capturing Business Strategy and Value in Enterprise Architecture to Support Portfolio Valuation. In *2012 16th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2012)*, pages 11–20.

IFIP-IFAC Task Force (1999). GERAM: Generalised Enterprise Reference Architecture and Methodology.

International Organization for Standardization (2000). ISO 15704:2000 Industrial Automation Systems - Requirements for enterprise-reference architectures and methodologies.

International Organization for Standardization (ISO) (2012). ISO/IEC 19507:2012 Information technology - Object Management Group Object Constraint Language (OCL).

Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The unified software development process.* The Addison-Wesley object technology series. Addison-Wesley, Reading, Mass.

Jadhav, A. and Sonar, R. (2009a). Analytic Hierarchy Process (AHP), Weighted Scoring Method (WSM), and Hybrid Knowledge Based System (HKBS) for Software Selection: A Comparative Study. In *Second International Conference on Emerging Trends in Engineering & Technology*, pages 991–997.

Jadhav, A. S. and Sonar, R. M. (2009b). Evaluating and selecting software packages: A review. *Information and Software Technology*, 51(3):555–563.

Jungnickel, D. (2013). *Graphs, networks, and algorithms*, volume 5 of *Algorithms and computation in mathematics*. Springer, Berlin, Heidelberg, 4 edition.

Keller, W. (2007). *IT-Unternehmensarchitektur: Von der Geschäftsstrategie zur optimalen IT-Unterstützung*. dpunkt.verlag, Heidelberg, 1 edition.

Kerzner, H. (2013). *Project management: A systems approach to planning, scheduling, and controlling*. John Wiley & Sons, Hoboken, New Jersey, 11 edition.

Kremen, P., Smid, M., and Kouba, Z. (2011). OWLDiff: A Practical Tool for Comparison and Merge of OWL Ontologies. In *22nd International Workshop on Database and Expert Systems Applications*, pages 229–233. IEEE.

Lautenbacher, F. (2010). *Semantic business process modeling: Principles, design support and realization*. Shaker Verlag, Aachen.

Lautenbacher, F., Diefenthaler, P., Langermeier, M., Mykhashchuk, M., and Bauer, B. (2013). Planning Support for Enterprise Changes. In Grabis, J., Kirikova, M., Zdravkovic, J., and Stirna, J., editors, *The Practice of Enterprise Modeling*, volume 165 of *Lecture Notes in Business Information Processing*, pages 54–68. Springer, Berlin, Heidelberg.

Loshin, D. (2009). *Master data management*. Elsevier/Morgan Kaufmann, Amsterdam and Boston.

Manola, F., Miller, E., and McBride, B. (2004). *RDF Primer*. World Wide Web Consortium.

Matthes, D. (2011). Enterprise Architecture Frameworks Kompendium: Über 50 Rahmenwerke für das IT-Management. *Enterprise Architecture Frameworks Kompendium*.

Matthes, F., Buckl, S., Leitel, J., and Schweda, C. M. (2008). *Enterprise architecture management tool survey 2008*. Technische Universität München, München.

Miller, J. G. (1978). *Living systems*. McGraw-Hill, New York.

Motik, B., Patel-Schneider, P. F., and Horrocks, I. (2009). OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax.

Murer, S., Bonati, B., and Furrer, F. J. (2011). *Managed evolution: A Strategy for very large information systems*. Springer, Berlin and New York.

Närman, P., Sommestad, T., Sandgren, S., and Ekstedt, M. (2009). A framework for assessing the cost of IT investments. In Kocaoglu, D. F., Anderson, T. R., and Daim, T. U., editors, *Portland International Center for Management of Engineering and Technology*, pages 3154–3166.

Natschläger, C. (2011). Towards a BPMN 2.0 Ontology. In Aalst, W., Mylopoulos, J., Rosemann, M., Shaw, M. J., Szyperski, C., Dijkman, R., Hofstetter, J., and Koehler, J., editors, *Business Process Model and Notation*, volume 95 of *Lecture Notes in Business Information Processing*, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg.

Niemann, K. D. (2006). *From enterprise architecture to IT governance: Elements of effective IT management*. Vieweg, Wiesbaden.

Object Management Group (2009). Ontology Definition Metamodel.

OConchuir, D. (2011). *Overview of the PMBOK guide: Shortcuts for PMP certification*. Springer, Berlin, 2 edition.

Op 't Land, M. (2008). *Applying architecture and ontology to the splitting and allying of enterprises*. PhD thesis, Technische Universiteit Delft, Delft, Netherlands.

Op 't Land, M., Proper, E., Waage, M., Cloo, J., and Steghuis, C. (2009). *Enterprise Architecture: Creating Value by Informed Governance*. The Enterprise Engineering Series. Springer Berlin Heidelberg, Berlin and Heidelberg.

Ortmann, J., Diefenthaler, P., Lautenbacher, F., Hess, C., and Chen, W. (2014). Unternehmensarchitekturen mit semantischen Technologien. *HMD Praxis der Wirtschaftsinformatik*, 51(5):616–626.

Österlind, M., Johnson, P., Karnati, K., Lagerstrom, R., and Valja, M. (2013). Enterprise Architecture Evaluation Using Utility Theory. In *2013 17th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW)*, pages 347–351.

Peffers, K., Tuunanen, T., Gengler, C., Rossi, M., Hui, W., Virtanen, V., and Bragge, J. (2006). The Design Science Research Process: A Model for Producing and Presenting Information Systems Research. In Chatterjee, S. and Hevner, A. R., editors, *Proceedings of the First International Conference on Design Science Research in Information Systems and Technology (DESRIST 2006)*, pages 83–106.

Postina, M. (2012). *Evolutionsmanagement prozess- und serviceorientierter Unternehmensarchitekturen*. PhD thesis, Edewecht and Oldenburg.

Postina, M., Sechyn, I., and Steffens, U. (2009). Gap analysis of application landscapes. In *2009 13th Enterprise Distributed Object Computing Conference Workshops*, pages 274–281. IEEE.

Project Management Institute (2008). *A guide to the project management body of knowledge (PMBOK Guide)*. Project Management Institute, Newtown Square, Pa., 4 edition.

Prud'hommeaux, E. and Seaborne, A. (2008). *SPARQL Query Language for RDF*. World Wide Web Consortium.

Pulkkinen, M. (2006). Systemic Management of Architectural Decisions in Enterprise Architecture Planning. Four Dimensions and Three Abstraction Levels. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, page 179a. IEEE.

Raymond, K. and Armstrong, L. (1995). *Open distributed processing: Experiences with distributed environments: proceedings of the third IFIP TC/WG 6.1 international conference on open distributed processing, 1994*. Chapman & Hall.

Rensink, A. (2004). Representing First-Order Logic Using Graphs. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Ehrig, H., Engels, G., Parisi-Presicce, F., and Rozenberg, G., editors, *Graph Transformations*, volume 3256 of *Lecture Notes in Computer Science*, pages 319–335. Springer Berlin Heidelberg, Berlin, Heidelberg.

Rensink, A. (2010). The Edge of Graph Transformation — Graphs for Behavioural Specification. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Engels, G., Lewerentz, C., Schäfer, W., Schürr, A., and Westfechtel, B., editors, *Graph Transformations and Model-Driven Engineering*, volume 5765 of *Lecture Notes in Computer Science*, pages 6–32. Springer Berlin Heidelberg, Berlin, Heidelberg.

Rensink, A. and Zambon, E. (2012). Pattern-Based Graph Abstraction. In Ehrig, H., Engels, G., Kreowski, H.-J., and Rozenberg, G., editors, *Graph Transformations*, volume 7562 of *Lecture Notes in Computer Science*, pages 66–80. Springer, Berlin, Heidelberg.

Rozenberg, G., editor (1997). *Handbook of graph grammars and computing by graph transformation: Foundations*, volume 1. World Scientific, River Edge, NJ, USA.

Russell, S. J. and Norvig, P. (2010). *Artificial intelligence: A modern approach*. Prentice Hall, Upper Saddle River, 3 edition.

Saaty, T. L. (1980). *The analytic hierarchy process: Planning, priority setting, resource allocation*. McGraw-Hill International Book Co., New York and London.

Schattenberg, B. (2009). *Hybrid planning & Scheduling*. PhD thesis, University Ulm.

Scheer, A.-W. (1991). *Architektur integrierter Informationssysteme: Grundlagen der Unternehmensmodellierung*. Springer-Verlag, Berlin and New York.

Schönherr, M. (2009). Towards a Common Terminology in the Discipline of Enterprise Architecture. In Feuerlicht, G. and Lamersdorf, W., editors, *Service-oriented computing, ICSOC 2008 workshops*, volume 5472 of *Lecture Notes in Computer Science*, pages 400–413. Springer, Berlin.

Shadbolt, N., Hall, W., and Berners-Lee, T. (2006). The Semantic Web Revisited. *Intelligent Systems*, 21(3):96–101.

Simon, D. (2009). Application landscape transformation and the role of Enterprise Architecture frameworks. In Steffens, U., editor, *MDD, SOA and IT-Management*. Gito, Berlin.

Sousa, P., Lima, J., Sampaio, A., and Pereira, C. (2009). An Approach for Creating and Managing Enterprise Blueprints: A Case for IT Blueprints. In Albani, A., Barjis, J., and Dietz, J. L. G., editors, *Advances in Enterprise Engineering III*, pages 70–84. Springer, Heidelberg.

Sousa, S., Marosin, D., Gaaloul, K., and Meyer, N. (2013). Assessing Risks and Opportunities in Enterprise Architecture using an extended ADT approach. In Gasevic, D., Hatala, M., Mothari Nezhad, H. D., and Reichert, M., editors, *Proceedings of the 17th IEEE International Enterprise Distributed Object Computing Conference*, pages 81–90.

Sowa, J. F. and Zachman, J. A. (1992). Extending and formalizing the framework for information systems architecture. *IBM Systems Journal*, 31(3):590–616.

Sterman, J. D. (1991). A Skeptic's Guide to Computer Models. In Barney, G. O., Kreutzer, W. B., and Garrett, M. J., editors, *Managing a nation*, pages 209–229. Westview Press, Boulder.

Sure, Y., Staab, S., and Studer, R. (2004). On-To-Knowledge Methodology (OTKM). In Staab, S. and Studer, R., editors, *Handbook on Ontologies*, pages 117–132. Springer Berlin Heidelberg, Berlin, Heidelberg.

Teece, D. J. (2007). Explicating dynamic capabilities: the nature and microfoundations of (sustainable) enterprise performance. *Strategic Management Journal*, 28(13):1319–1350.

The Open Group (2011). *TOGAF Version 9.1*. TOGAF series. Van Haren Publishing, Zaltbommel, 10 edition.

The Open Group (2012). *Archimate 2.0 Specification*. Van Haren Publishing.

Triantaphyllou, E. (2000). *Multi-criteria decision making methods: A comparative study*, volume v. 44 of *Applied optimization*. Kluwer Academic Publishers, Dordrecht and Boston, Mass.

Vaquero, T. S., Silva, J. R., and Beck, C. J. (2011). A Brief Review of Tools and Methods for Knowledge Engineering for Planning & Scheduling. In Bartak, R., Fratini, S., McCluskey, L., and Vaquero, T. S., editors, *KEPS 2011 Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling*, pages 7–14.

Wegmann, A. (2003). On the Systemic Enterprise Architecture Methodology (Seam). In *Proceedings of the 5th International Conference on Enterprise Information Systems*, pages 483–490.

Winter, R. and Fischer, R. (2006). Essential Layers, Artifacts, and Dependencies of Enterprise Architecture. In *10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW'06)*, page 30. IEEE.

Winter, R., Gericke, A., and Bucher, T. (2009). Method Versus Model – Two Sides of the Same Coin? In Albani, A., Barjis, J., and Dietz, J. L. G., editors, *Advances in Enterprise Engineering III*, pages 1–15. Springer, Heidelberg.

Zachman, J. A. (1987). A framework for information systems architecture. *IBM Systems Journal*, 26(3):276–292.

Zambon, E. (2013). *Abstract graph transformation — Theory and Practice*. Wöhrmann Print Service.

Zhu, H. (2005). *Software design methodology*. Elsevier Butterworth-Heinemann, Amsterdam and Boston.

Zimmermann, S. (2008). IT-Portfoliomanagement – Ein Konzept zur Bewertung und Gestaltung von IT. *Informatik-Spektrum*, 31(5):460–468.

List of Tables

List of Figures

Listings

Acronyms

Acronym	Term
ADM	Architecture Development Method
AHP	Analytic Hierarchy Process
ATO	Action-Threat-Opportunities
BPMN	Business Process Modeling and Notation
c. f.	confer
DMDM	Development Master Data Management
DSIG	Data Signature
EA	Enterprise Architecture
EAG	Enterprise Architecture Graph
EAM	Enterprise Architecture Management
EAMS	Enterprise Architecture Management System
EATG	Enterprise Architecture Type Graph
e. g.	exempli gratia
et al.	et alii
ff.	and the following
GG	Graph Grammar
GROOVE	GRaphs for Object-Oriented VErification
GTS	Graph Transformation System
GXL	Graph Exchange Language
IT	Information Technology
LHS	Left Hand Side
MCDM	Multi-Criteria Decision Making
MODM	Multi-Objective Decision Making
NAC	Nested Application Condition
OFFIS	Institute for Information Technology of the University of Oldenburg
OWL	Web Ontology Language
p.	page
PDM	Planning Domain Model
RDF	Resource Description Framework
RHS	Right Hand Side
RM-ODP	Reference Model for Open Distributed Processing
SEAM-Tool	Serviceoriented EAM-Tool
SPARQL	SPARQL Protocol And RDF Query Language
TOGAF	The Open Group Architecture Framework
UML	Unified Modeling Language
WPM	Weighted Product Model
WSM	Weighted Sum Model
XML	Extensible Markup Language

A LHS, RHS, and NAC for Transformation Actions

A.1 Target Architecture Selection - Bottom-Up Transformation Actions

$P_{ReplaceAS}$

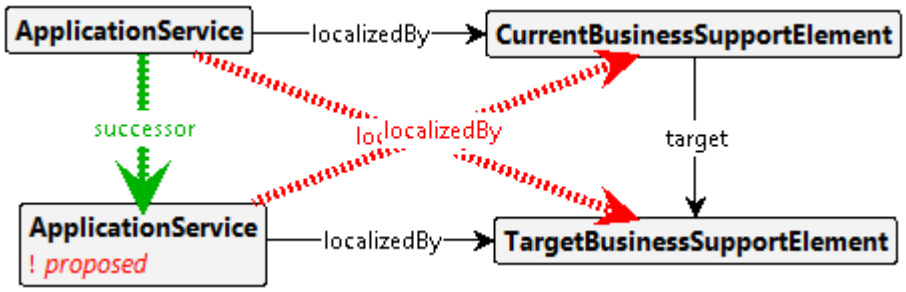

Figure A.1: LHS, RHS, and NAC of $P_{ReplaceAS}$

$P_{ReuseAS}$

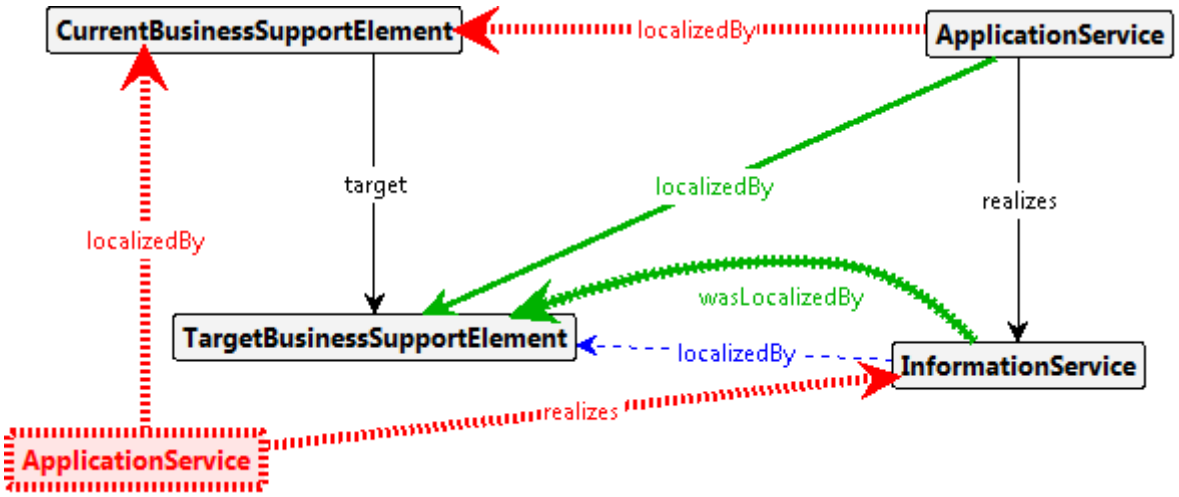

Figure A.2: LHS, RHS, and NAC of $P_{ReuseAS}$

$P_{EnhanceAS_{BU}}$

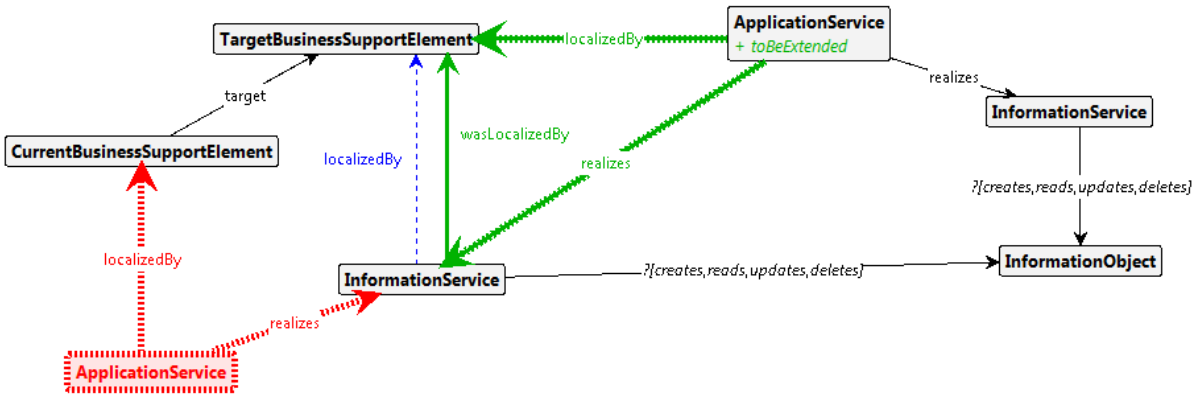

Figure A.3: LHS, RHS, and NAC of $P_{EnhanceAS_{BU}}$

$P_{NewASOldA}$

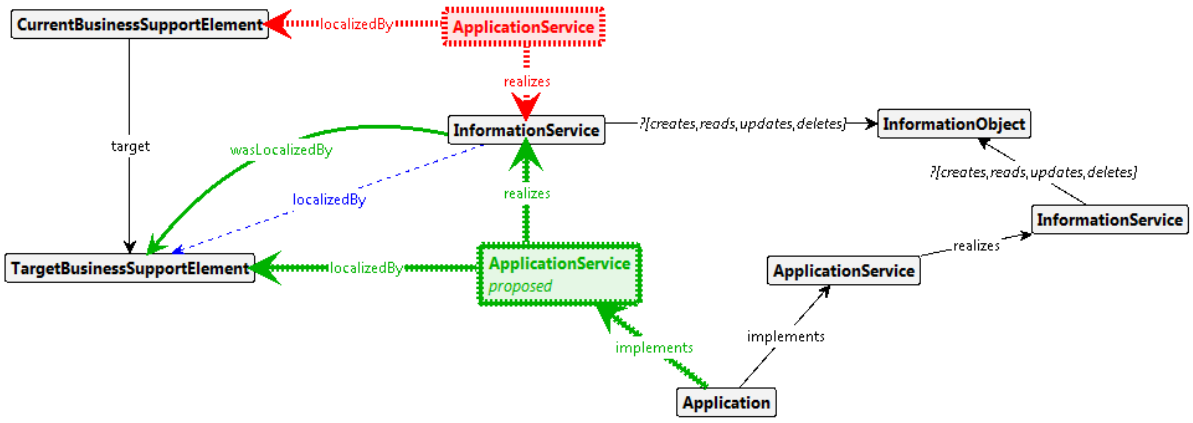

Figure A.4: LHS, RHS, and NAC of $P_{NewASOldA}$

$P_{NewASNewA}$

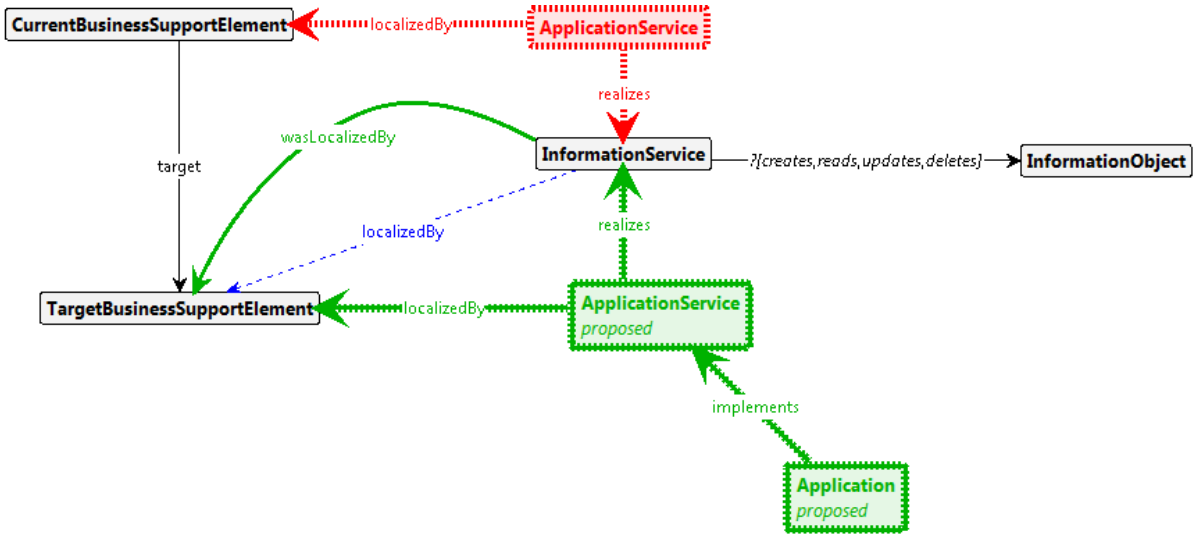

Figure A.5: LHS, RHS, and NAC of $P_{NewASNewA}$

$P_{ReuseAS_{IS}}$

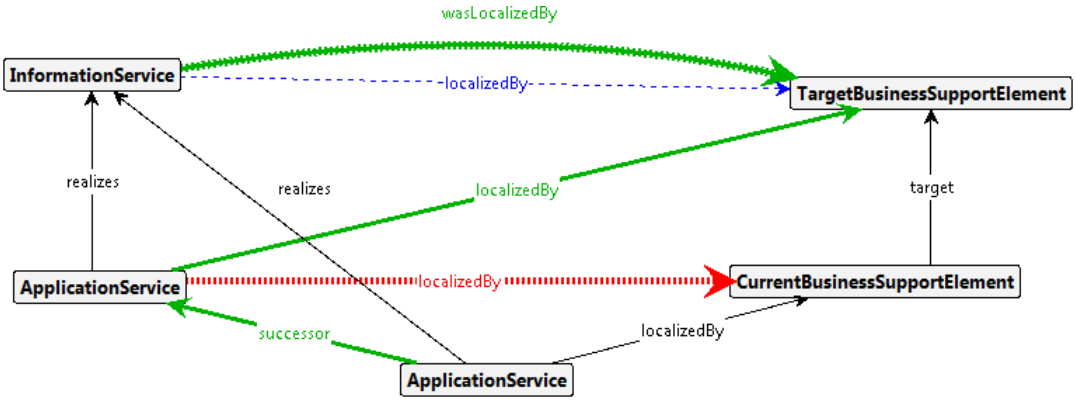

Figure A.6: LHS, RHS, and NAC of $P_{ReuseAS_{IS}}$

$P_{EnhanceAS_{IS}}$

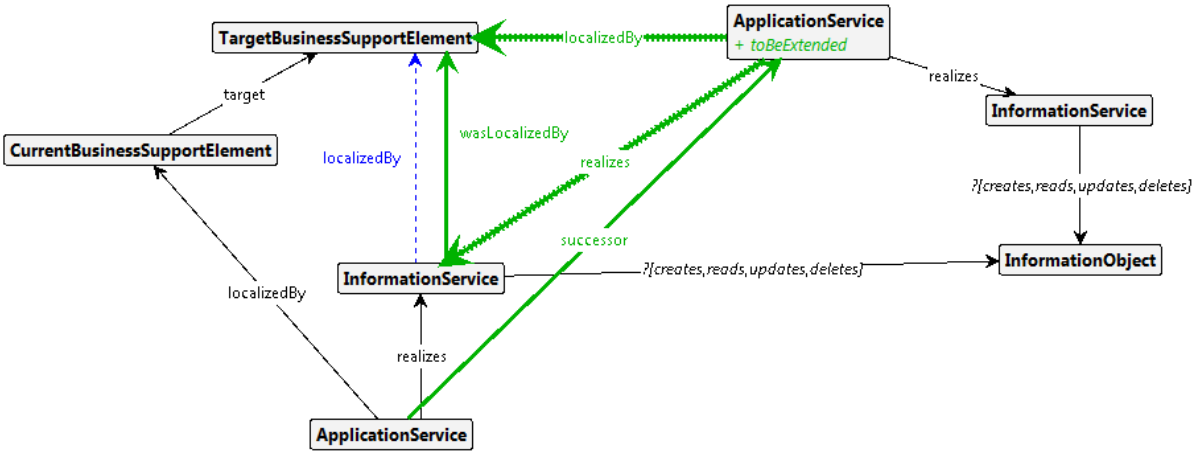

Figure A.7: LHS, RHS, and NAC of $P_{EnhanceAS_{IS}}$

$P_{NewASOldA_{IS}}$

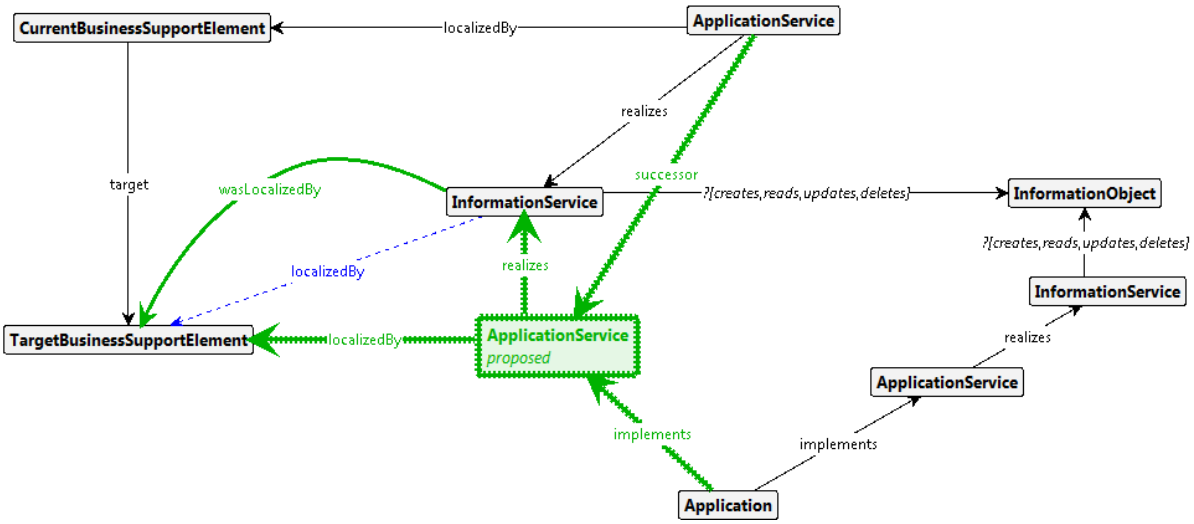

Figure A.8: LHS, RHS, and NAC of $P_{NewASOldA_{IS}}$

$P_{NewASNewA_{IS}}$

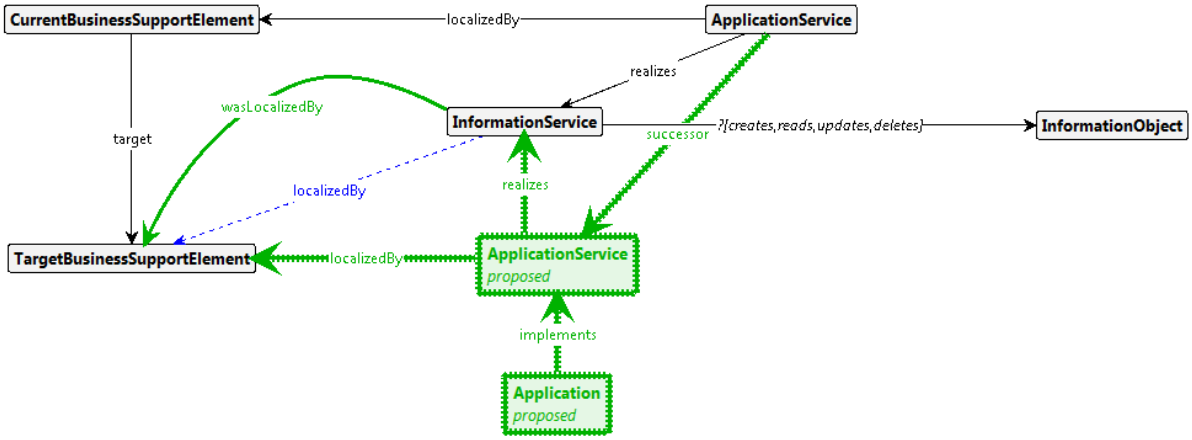

Figure A.9: LHS, RHS, and NAC of $P_{NewASNewA_{IS}}$

A.2 Target Architecture Selection - Top-Down Transformation Actions and Suggestions

P_{NewApp}

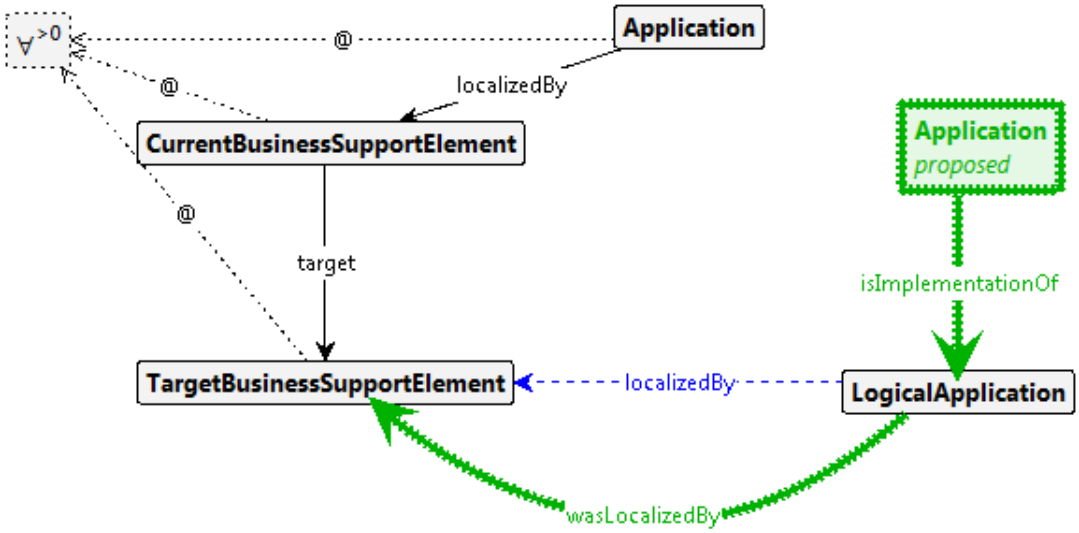

Figure A.10: LHS, RHS, and NAC of P_{NewApp}

$P_{ReplaceApp}$

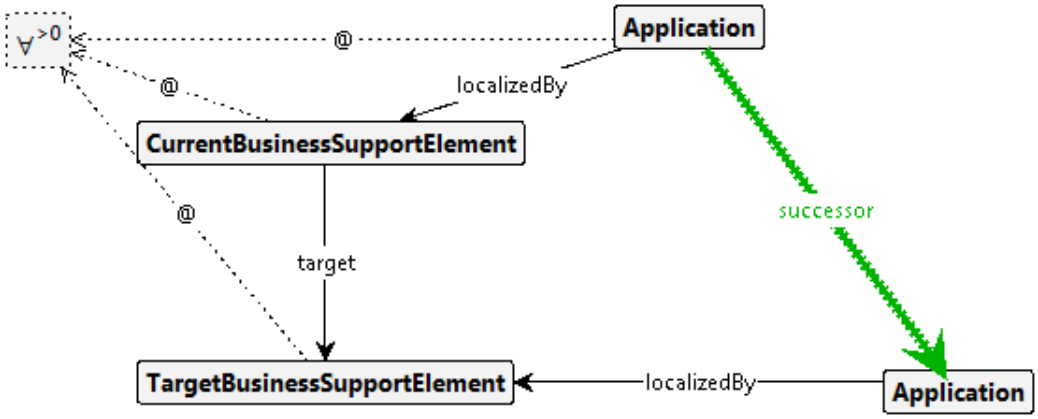

Figure A.11: LHS, RHS, and NAC of $P_{ReplaceApp}$

$P_{ReuseApp}$

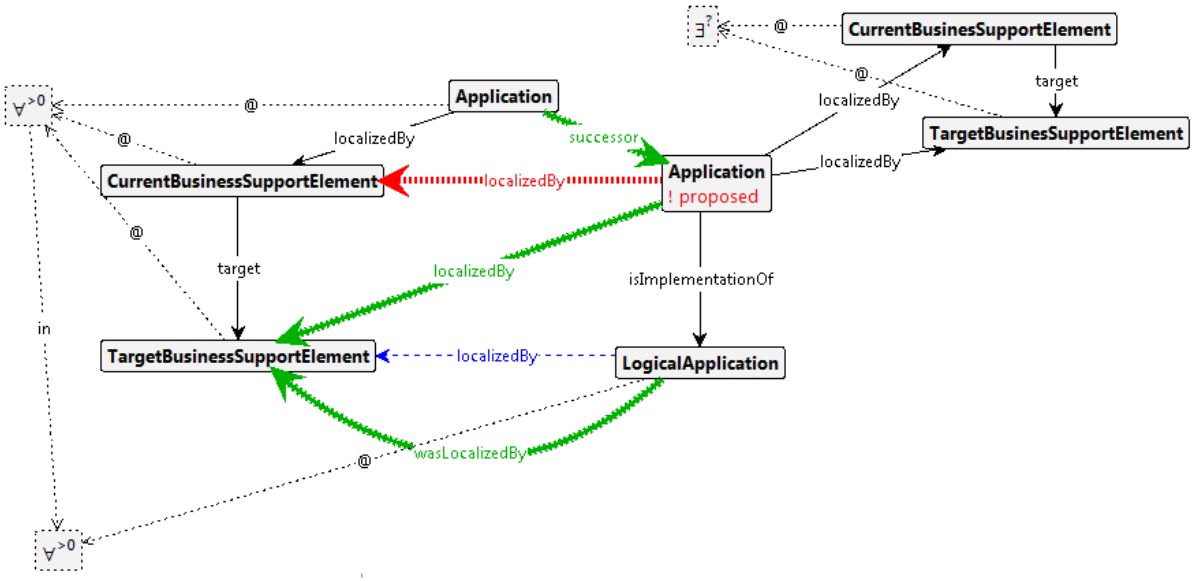

Figure A.12: *LHS*, *RHS*, and *NAC* of $P_{ReuseApp}$

$P_{EnhanceAS_{TD}}$

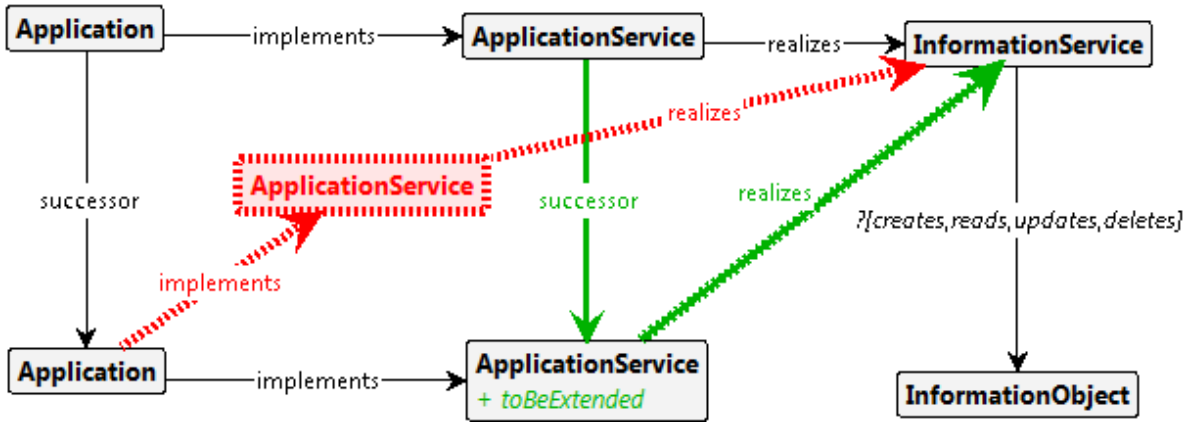

Figure A.13: *LHS*, *RHS*, and *NAC* of $P_{EnhanceAS_{TD}}$

$P_{NewAS_{TD}}$

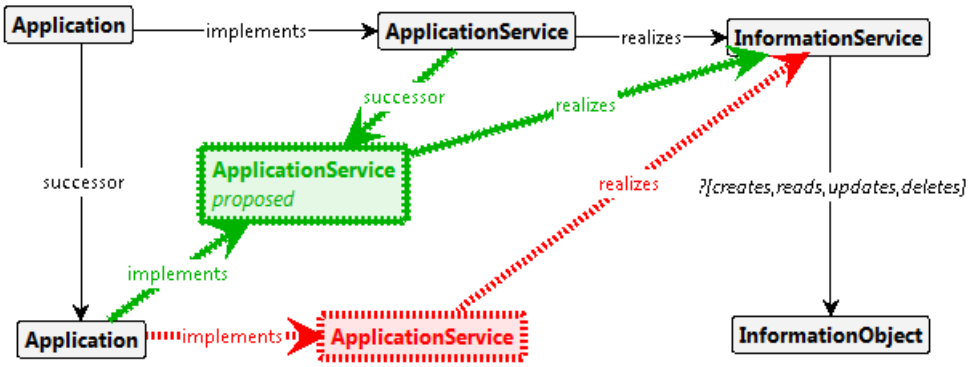

Figure A.14: LHS, RHS, and NAC of $P_{NewAS_{TD}}$

A.3 Transformation Path Generation - Default Transformation Actions

$P_{DevelopApp}$

Figure A.15: LHS, RHS, and NAC of $P_{DevelopApp}$

$P_{DevelopAS}$

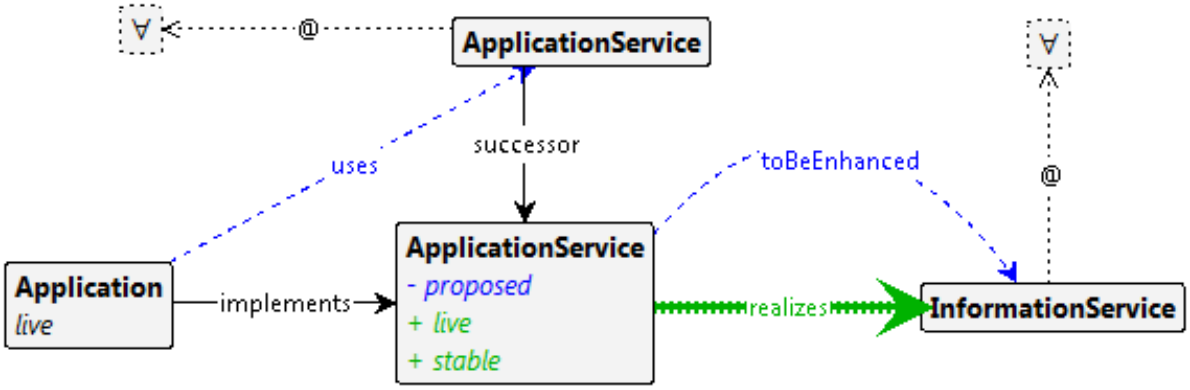

Figure A.16: LHS, RHS, and NAC of $P_{DevelopAS}$

$P_{ChangeDepForSuccessor}$

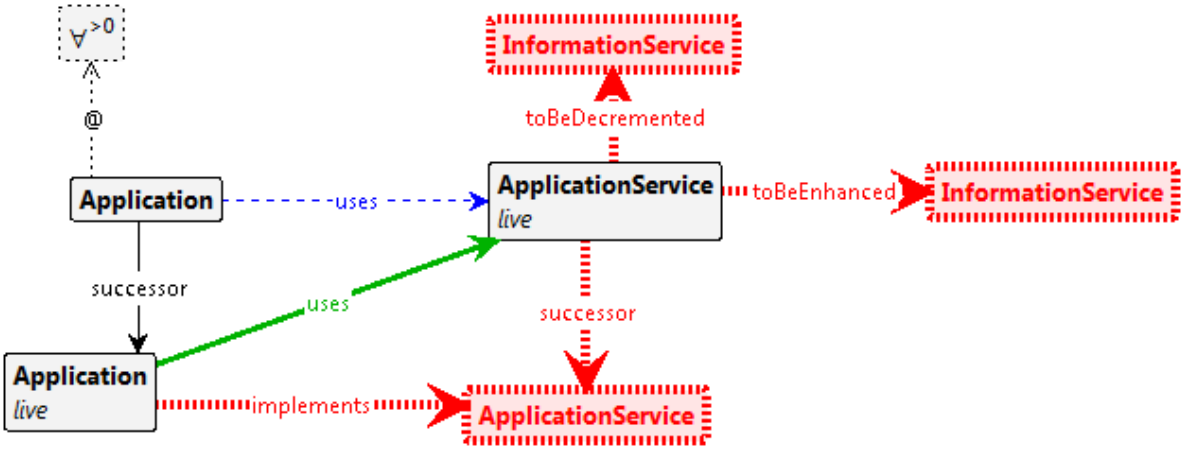

Figure A.17: LHS, RHS, and NAC of $P_{ChangeDepForSuccessor}$

$P_{ChangeDepToSuccessorAS}$

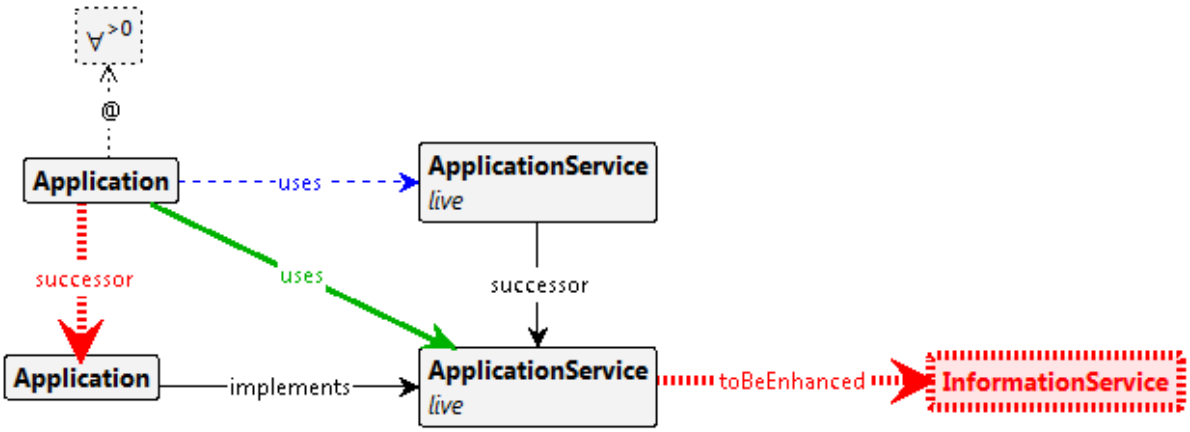

Figure A.18: LHS, RHS, and NAC of $P_{ChangeDepToSuccessorAS}$

$P_{ShutdownAS_{noSucc}}$

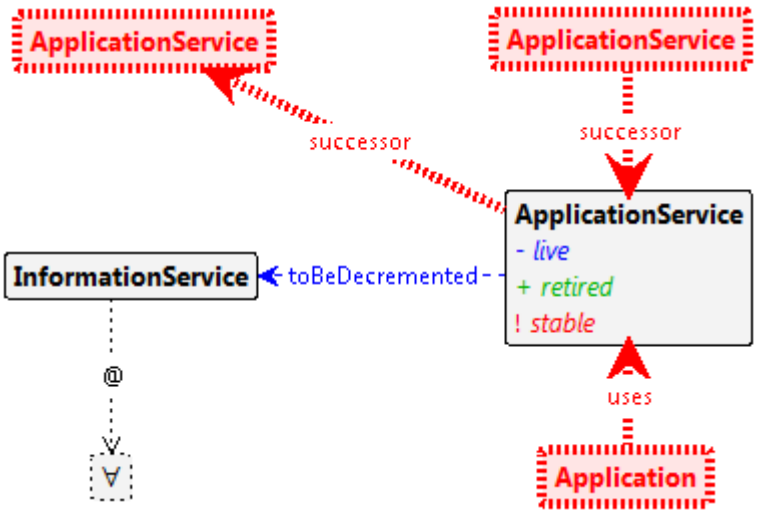

Figure A.19: LHS, RHS, and NAC of $P_{ShutdownAS_{noSucc}}$

$P_{ShutdownAS_{succ}}$

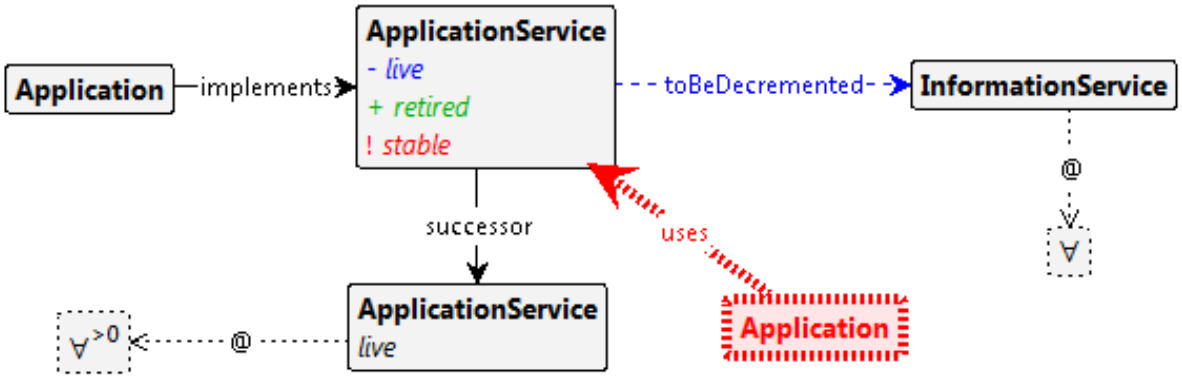

Figure A.20: LHS, RHS, and NAC of $P_{ShutdownAS_{succ}}$

$P_{ShutdownAS_{allSucc}}$

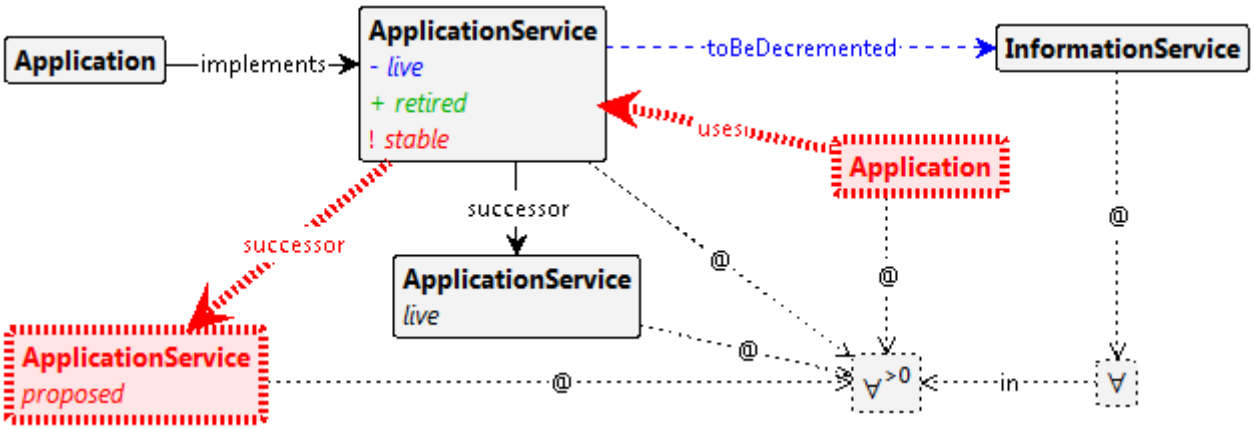

Figure A.21: LHS, RHS, and NAC of $P_{ShutdownAS_{allSucc}}$

$P_{ShutdownApp_{noSucc}}$

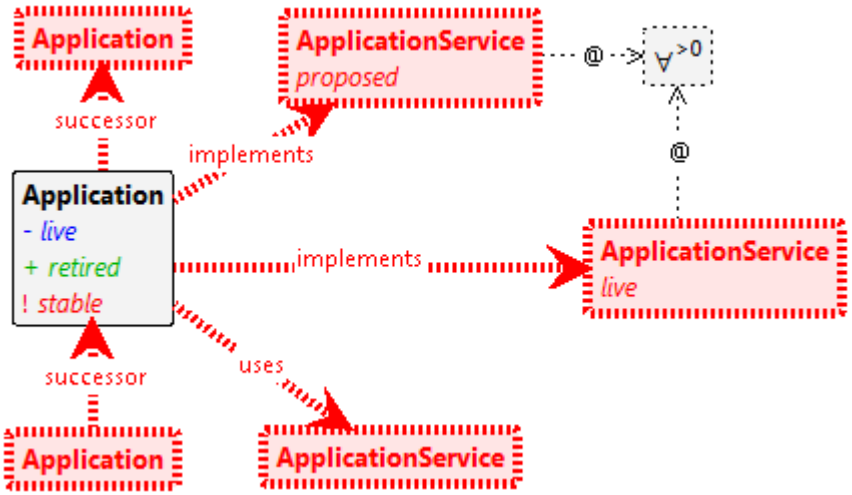

Figure A.22: LHS, RHS, and NAC of $P_{ShutdownApp_{noSucc}}$

$P_{ShutdownAS_{succ}}$

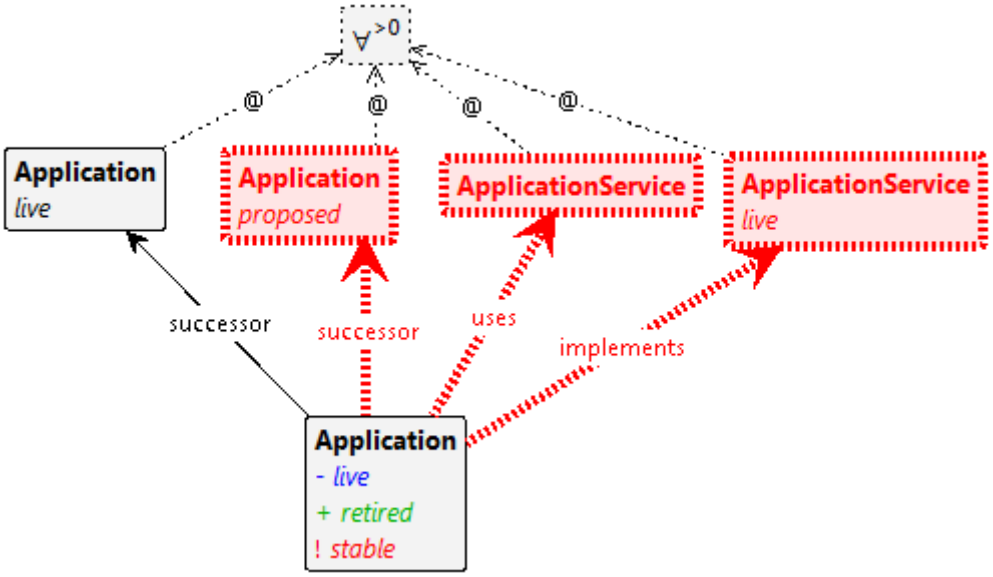

Figure A.23: LHS, RHS, and NAC of $P_{ShutdownApp_{succ}}$

$P_{ShutdownApp_{allSucc}}$

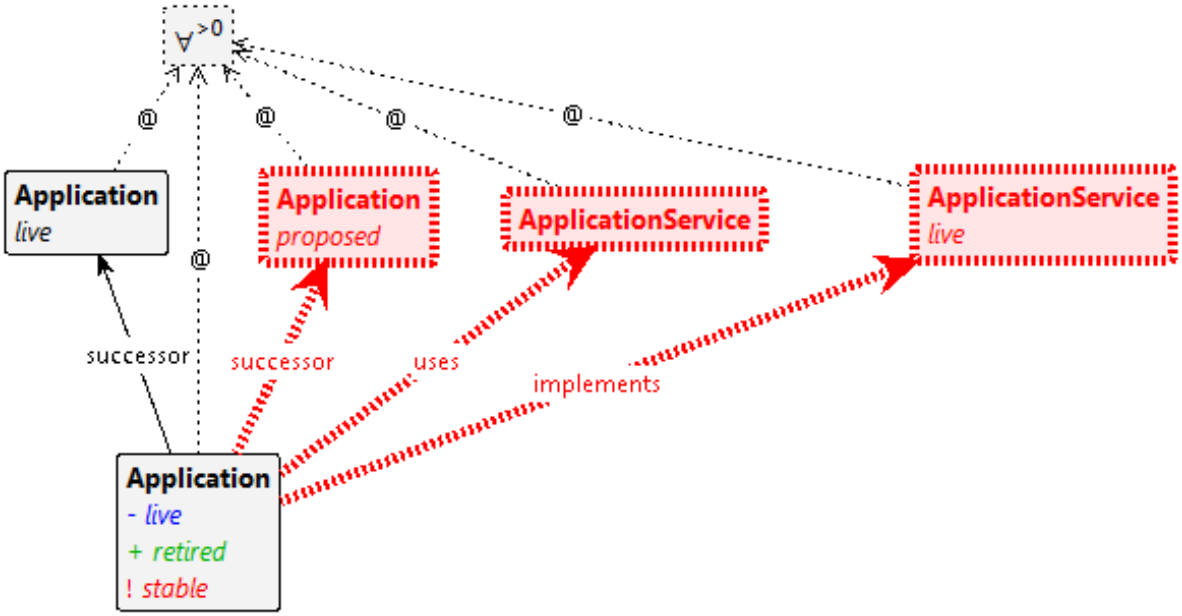

Figure A.24: LHS, RHS, and NAC of $P_{ShutdownApp_{allSucc}}$

$P_{ShutdownAS_{succ}}$

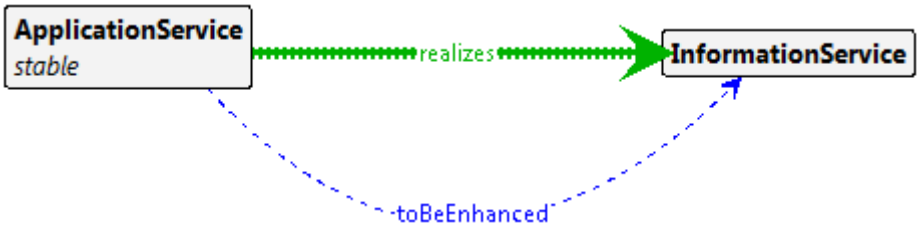

Figure A.25: LHS, RHS, and NAC of $P_{EnhanceStableAS}$

$P_{ShutdownApp_{allSucc}}$

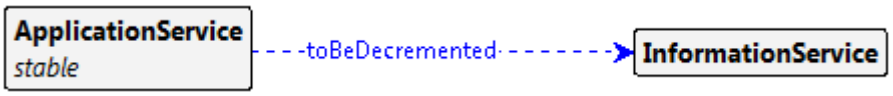

Figure A.26: LHS, RHS, and NAC of $P_{DecrementStableAS}$

A.4 Transformation Path Generation - Reference Scenarios

$P_{RS_{SS2}}$

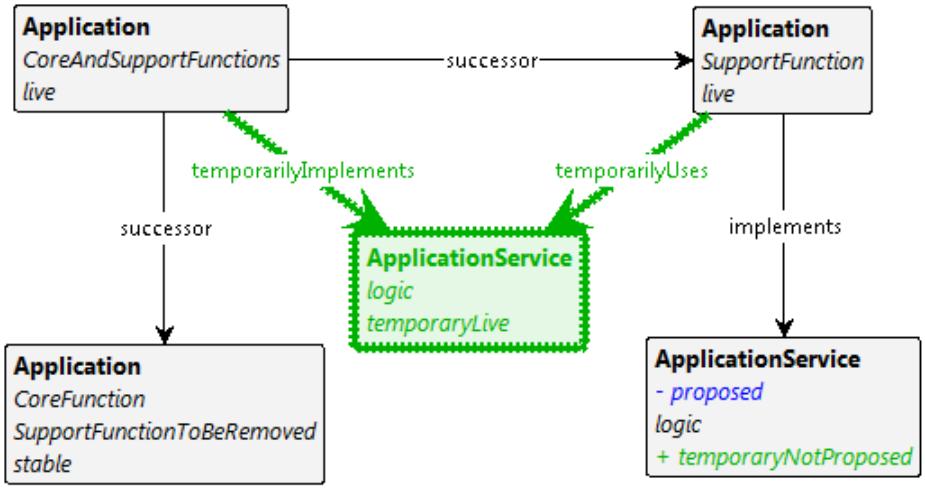

Figure A.27: LHS, RHS, and NAC of $P_{RS_{SS2}}$

$P_{RS_{SS3}}$

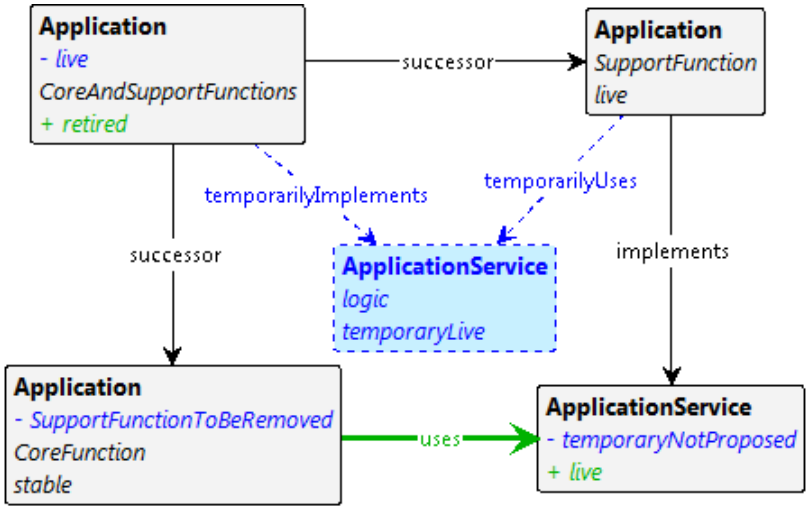

Figure A.28: LHS, RHS, and NAC of $P_{RS_{SS3}}$

$P_{RS_{IC2}}$

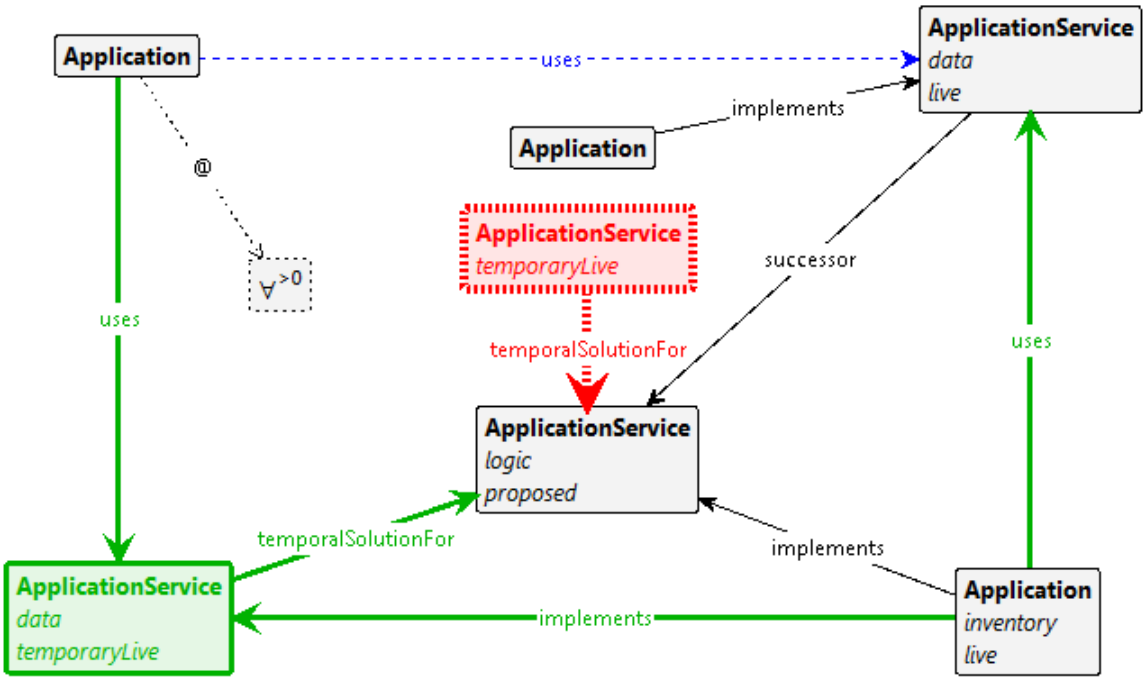

Figure A.29: LHS, RHS, and NAC of $P_{RS_{IC2}}$

$P_{RS_{IC3}}$

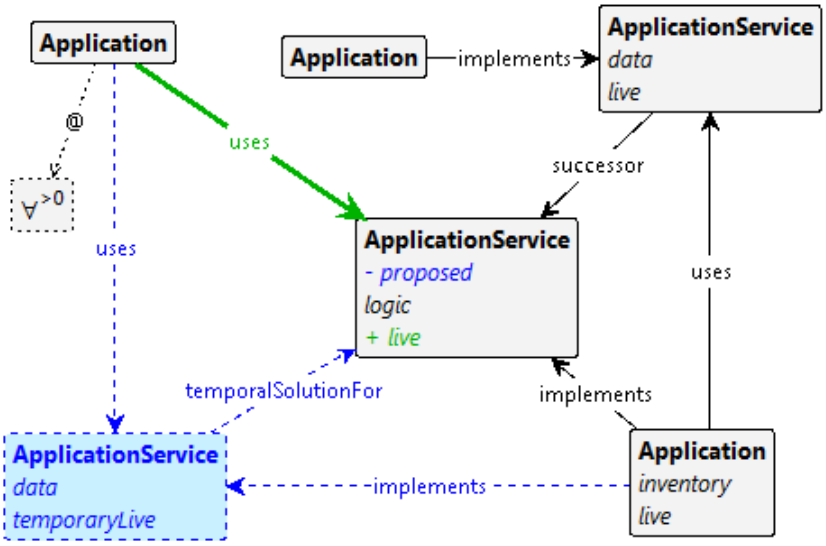

Figure A.30: LHS, RHS, and NAC of $P_{RS_{IC3}}$

$P_{RS_{IC4}}$

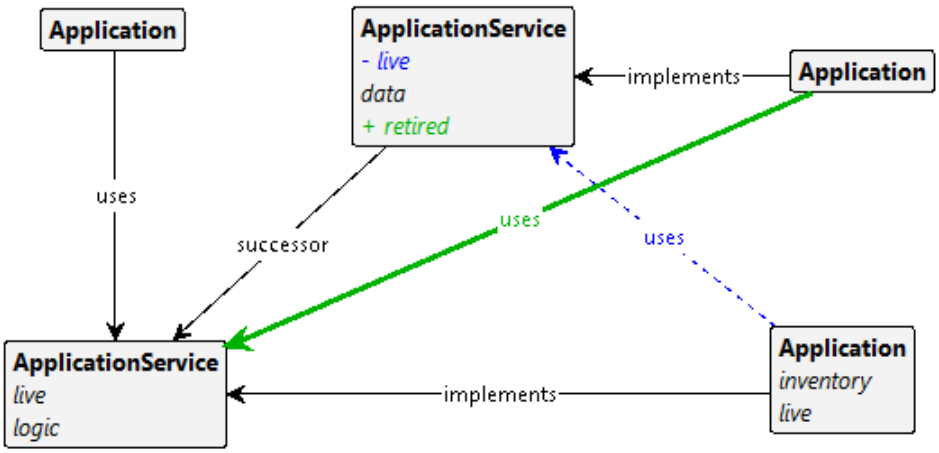

Figure A.31: LHS, RHS, and NAC of $P_{RS_{IC4}}$

$P_{RS_{DevelopAS}}$

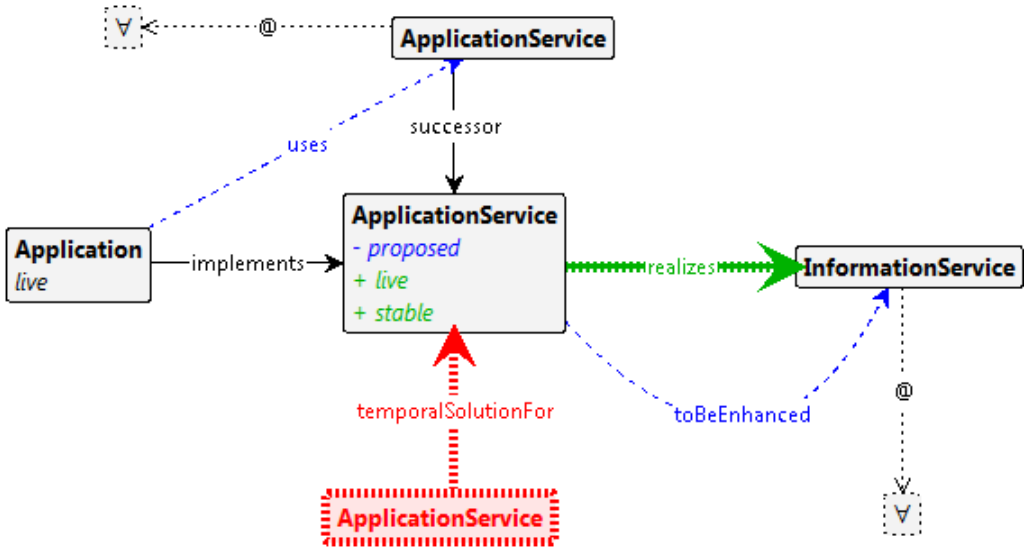

Figure A.32: LHS, RHS, and NAC of $P_{RS_{DevelopAS}}$

B LHS, RHS, and NAC for Typed Attributed Graph Productions in Bottom-Up and Top-Down Approach

B.1 Target Architecture Selection - Bottom-Up - Typed Attributed Graph Productions for Finalizing G_{goal}

$P_{finalize_{BU1}}$

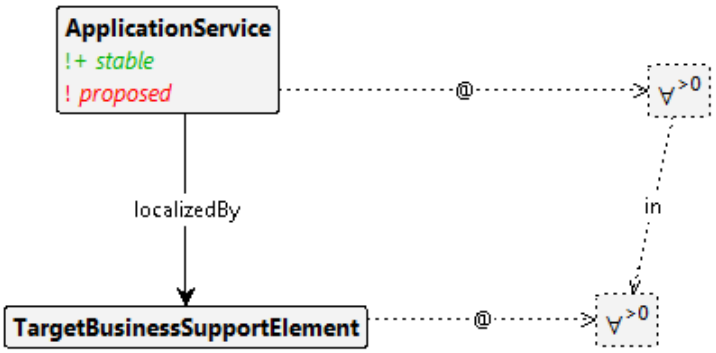

Figure B.1: LHS, RHS, and NAC of $P_{finalize_{BU1}}$

$P_{finalize_{BU2}}$

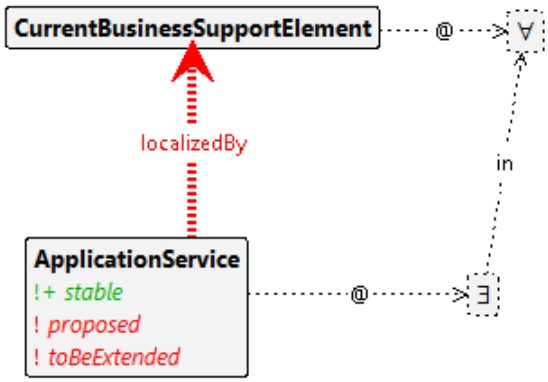

Figure B.2: LHS, RHS, and NAC of $P_{finalize_{BU2}}$

$P_{finalize_{BU3}}$

Figure B.3: LHS, RHS, and NAC of $P_{finalize_{BU3}}$

$P_{finalize_{BU4}}$

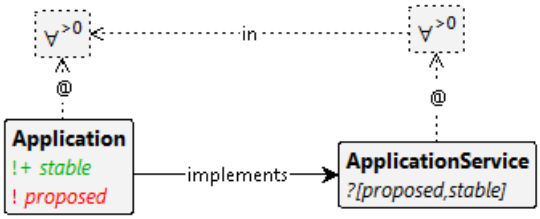

Figure B.4: LHS, RHS, and NAC of $P_{finalize_{BU4}}$

$P_{finalize_{BU5}}$

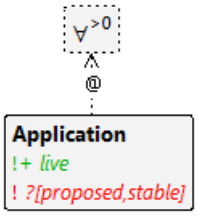

Figure B.5: LHS, RHS, and NAC of $P_{finalize_{BU5}}$

$P_{finalize_{BU6}}$

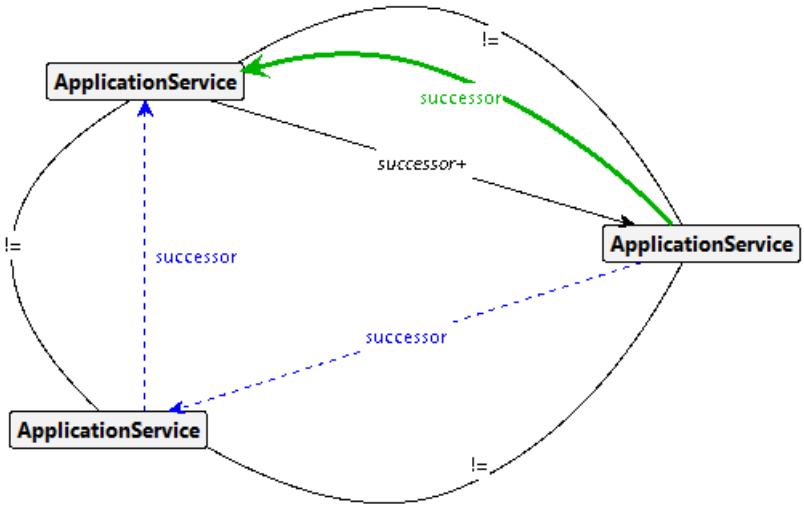

Figure B.6: LHS, RHS, and NAC of $P_{finalize_{BU6}}$

$P_{finalize_{BU7}}$

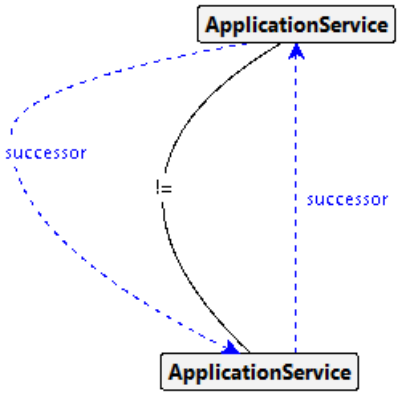

Figure B.7: LHS, RHS, and NAC of $P_{finalize_{BU7}}$

$P_{finalize_{BU8}}$

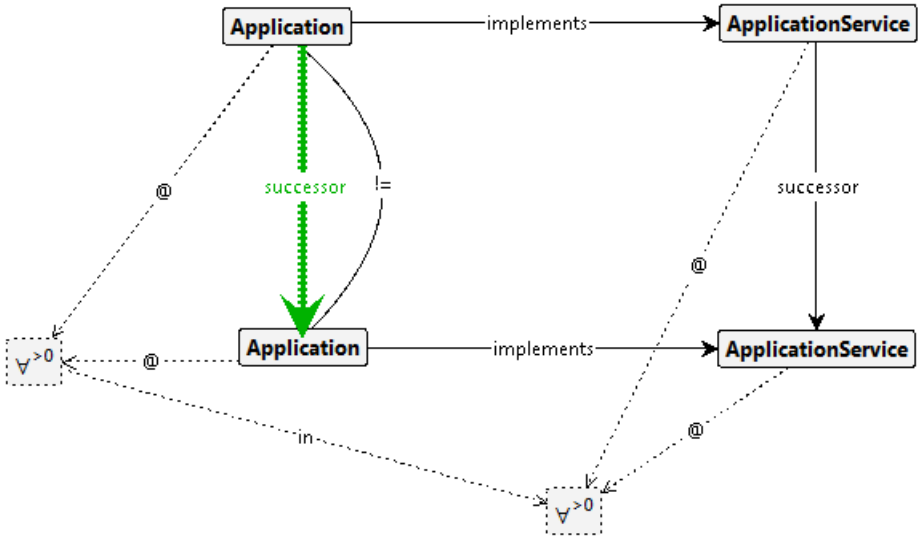

Figure B.8: LHS, RHS, and NAC of $P_{finalize_{BU8}}$

$P_{finalize_{BU9}}$

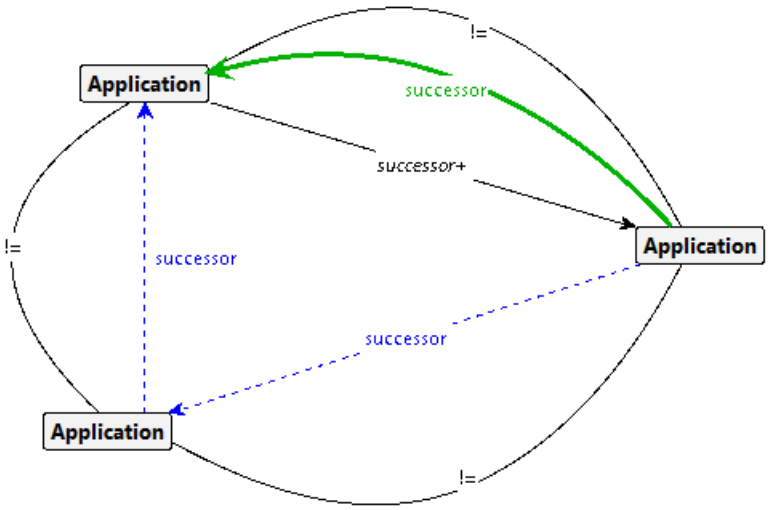

Figure B.9: LHS, RHS, and NAC of $P_{finalize_{BU9}}$

$P_{finalize_{BU10}}$

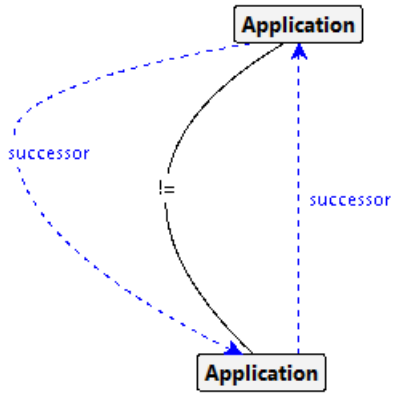

Figure B.10: LHS, RHS, and NAC of $P_{finalize_{BU10}}$

B.2 Target Architecture Selection - Top-Down - Typed Attributed Graph Production

$P_{mergeNewApps}$

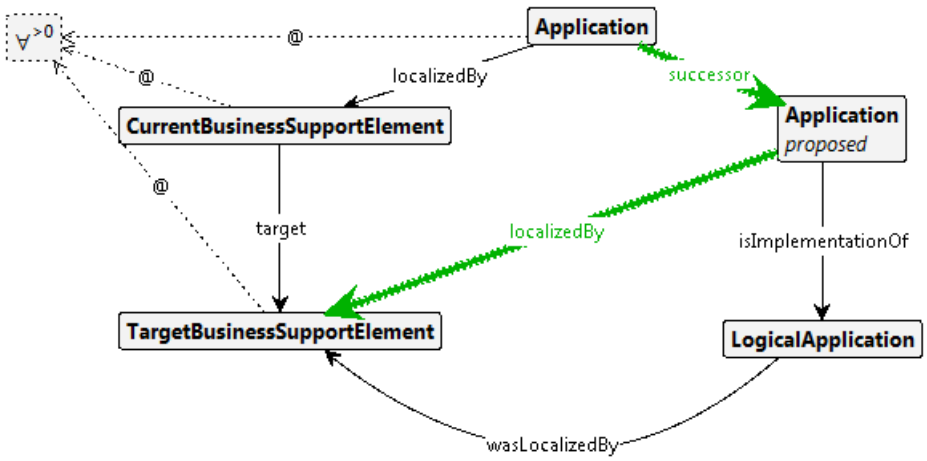

Figure B.11: LHS, RHS, and NAC of $P_{mergeNewApps}$

C Excerpt of the EAM Ontology

Listing C.1: Serialization of Ontology for Enterprise Architecture Management

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
    <!ENTITY lea "http://www.softplant.de/lea#" >
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<rdf:RDF xmlns="http://www.softplant.de/lea#"
     xml:base="http://www.softplant.de/lea"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:lea="http://www.softplant.de/lea#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://www.softplant.de/lea"/>


    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Annotation properties
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->

    <!-- http://www.softplant.de/lea#description -->

    <owl:AnnotationProperty rdf:about="&lea;description">
        <rdfs:label xml:lang="de">Beschreibung</rdfs:label>
    </owl:AnnotationProperty>


    <!-- http://www.softplant.de/lea#hatQuelle -->

    <owl:AnnotationProperty rdf:about="&lea;hatQuelle">
        <rdfs:label xml:lang="de">hat Quelle</rdfs:label>
    </owl:AnnotationProperty>


    <!-- http://www.softplant.de/lea#istBestandteilDerArchitekturebene -->

    <owl:AnnotationProperty rdf:about="&lea;istBestandteilDerArchitekturebene">
        <rdfs:label xml:lang="de">ist Bestandteil der Architekturebene</rdfs:label>
    </owl:AnnotationProperty>


    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
```

```
52      //
        // Object Properties
54      //
        //////////////////////////////////////////////////////////////////////////////////
56       -->

58      <!-- http://www.softplant.de/lea#AnwendungHatNachfolger -->

60      <owl:ObjectProperty rdf:about="&lea;AnwendungHatNachfolger">
            <rdfs:label xml:lang="de">hat Nachfolger</rdfs:label>
62          <rdfs:label xml:lang="en">has sucessor</rdfs:label>
            <rdfs:range rdf:resource="&lea;Anwendung"/>
64          <rdfs:domain rdf:resource="&lea;Anwendung"/>
        </owl:ObjectProperty>
66

68      <!-- http://www.softplant.de/lea#AnwendungImplementiertAnwendungsservice -->

70      <owl:ObjectProperty rdf:about="&lea;AnwendungImplementiertAnwendungsservice">
            <rdfs:label xml:lang="de">implementiert</rdfs:label>
72          <rdfs:label xml:lang="en">implements</rdfs:label>
            <rdfs:domain rdf:resource="&lea;Anwendung"/>
74          <rdfs:range rdf:resource="&lea;Anwendungsservice"/>
        </owl:ObjectProperty>
76

78      <!-- http://www.softplant.de/lea#AnwendungIstNachfolgerVonSichSelbst -->

80      <owl:ObjectProperty rdf:about="&lea;AnwendungIstNachfolgerVonSichSelbst">
            <rdfs:label xml:lang="de">ist Nachfolger von sich selbst</rdfs:label>
82          <rdfs:label xml:lang="en">is successor of itself</rdfs:label>
            <rdfs:range rdf:resource="&lea;Anwendung"/>
84          <rdfs:domain rdf:resource="&lea;Anwendung"/>
        </owl:ObjectProperty>
86

88      <!-- http://www.softplant.de/lea#AnwendungNutztAnwendungsservice -->

90      <owl:ObjectProperty rdf:about="&lea;AnwendungNutztAnwendungsservice">
            <rdfs:label xml:lang="de">nutzt</rdfs:label>
92          <rdfs:label xml:lang="en">uses</rdfs:label>
            <rdfs:domain rdf:resource="&lea;Anwendung"/>
94          <rdfs:range rdf:resource="&lea;Anwendungsservice"/>
        </owl:ObjectProperty>
96

98      <!-- http://www.softplant.de/lea#AnwendungsserviceHatNachfolger -->

100     <owl:ObjectProperty rdf:about="&lea;AnwendungsserviceHatNachfolger">
            <rdfs:label xml:lang="de">hat Nachfolger</rdfs:label>
102         <rdfs:label xml:lang="en">has successor</rdfs:label>
            <rdfs:range rdf:resource="&lea;Anwendungsservice"/>
104         <rdfs:domain rdf:resource="&lea;Anwendungsservice"/>
        </owl:ObjectProperty>
106

108     <!-- http://www.softplant.de/lea#AnwendungsserviceIstNachfolgerVonSichSelbst -->

110     <owl:ObjectProperty rdf:about="&lea;AnwendungsserviceIstNachfolgerVonSichSelbst">
            <rdfs:label xml:lang="de">ist Nachfolger von sich selbst</rdfs:label>
112         <rdfs:label xml:lang="en">is successor of itself</rdfs:label>
            <rdfs:range rdf:resource="&lea;Anwendungsservice"/>
114         <rdfs:domain rdf:resource="&lea;Anwendungsservice"/>
        </owl:ObjectProperty>
116
```

```
118     <!-- http://www.softplant.de/lea#AnwendungsserviceRealisiertInformationsservice -->

120     <owl:ObjectProperty rdf:about="&lea;AnwendungsserviceRealisiertInformationsservice">
            <rdfs:label xml:lang="de">realisiert</rdfs:label>
122         <rdfs:label xml:lang="en">realizes</rdfs:label>
            <rdfs:domain rdf:resource="&lea;Anwendungsservice"/>
124         <rdfs:range rdf:resource="&lea;Informationsservice"/>
        </owl:ObjectProperty>
126

128     <!-- http://www.softplant.de/lea#BebauungselementBebautProzessaktivitaet -->

130     <owl:ObjectProperty rdf:about="&lea;BebauungselementBebautProzessaktivitaet">
            <rdfs:label xml:lang="de">bebaut durch</rdfs:label>
132         <rdfs:label xml:lang="en">localized by</rdfs:label>
            <rdfs:domain rdf:resource="&lea;Bebauungselement"/>
134         <rdfs:range rdf:resource="&lea;Prozessaktivitaet"/>
        </owl:ObjectProperty>
136

138     <!-- http://www.softplant.de/lea#BebauungselementImIstBebautAnwendungsservice -->

140     <owl:ObjectProperty rdf:about="&lea;BebauungselementImIstBebautAnwendungsservice">
            <rdfs:label xml:lang="de">bebaut</rdfs:label>
142         <rdfs:label xml:lang="en">localizes</rdfs:label>
            <rdfs:range rdf:resource="&lea;Anwendungsservice"/>
144         <rdfs:domain rdf:resource="&lea;BebauungselementImIst"/>
        </owl:ObjectProperty>
146

148     <!-- http://www.softplant.de/lea#BebauungselementImSollBebautAnwendungsservice -->

150     <owl:ObjectProperty rdf:about="&lea;BebauungselementImSollBebautAnwendungsservice">
            <rdfs:label xml:lang="de">bebaut</rdfs:label>
152         <rdfs:label xml:lang="en">localizes</rdfs:label>
            <rdfs:range rdf:resource="&lea;Anwendungsservice"/>
154         <rdfs:domain rdf:resource="&lea;BebauungselementImSoll"/>
        </owl:ObjectProperty>
156

158     <!-- http://www.softplant.de/lea#BebauungselementImSollBebautInformationsservice -->

160     <owl:ObjectProperty rdf:about="&lea;BebauungselementImSollBebautInformationsservice">
            <rdfs:label xml:lang="de">bebaut</rdfs:label>
162         <rdfs:label xml:lang="en">localizes</rdfs:label>
            <rdfs:domain rdf:resource="&lea;BebauungselementImSoll"/>
164         <rdfs:range rdf:resource="&lea;Informationsservice"/>
        </owl:ObjectProperty>
166

168     <!-- http://www.softplant.de/lea#InformationsserviceVerwendetInformationsobjekt -->

170     <owl:ObjectProperty rdf:about="&lea;InformationsserviceVerwendetInformationsobjekt">
            <rdfs:label xml:lang="de">verwendet</rdfs:label>
172         <rdfs:label xml:lang="en">uses</rdfs:label>
            <rdfs:range rdf:resource="&lea;Informationsobjekt"/>
174         <rdfs:domain rdf:resource="&lea;Informationsservice"/>
        </owl:ObjectProperty>
176


178
        <!-- http://www.softplant.de/lea#ProzessBestehtAusProzessaktivitaet -->
180
        <owl:ObjectProperty rdf:about="&lea;ProzessBestehtAusProzessaktivitaet">
182         <rdfs:label xml:lang="de">besteht aus</rdfs:label>
            <rdfs:label xml:lang="en">consists of</rdfs:label>
```

```
184         <rdfs:domain rdf:resource="&lea;Prozess"/>
            <rdfs:range rdf:resource="&lea;Prozessaktivitaet"/>
186     </owl:ObjectProperty>

188
        <!--
190     ///////////////////////////////////////////////////////////////////////////////////
        //
192     // Data properties
        //
194     ///////////////////////////////////////////////////////////////////////////////////
         -->
196
        <!-- http://www.softplant.de/lea#hatLebezyklusphase -->
198
        <owl:DatatypeProperty rdf:about="&lea;hatLebezyklusphase">
200         <rdfs:label xml:lang="de">hat Lebenszyklusphase</rdfs:label>
            <rdfs:label xml:lang="en">has lifecylce phase</rdfs:label>
202         <rdfs:domain rdf:resource="&lea;Anwendung"/>
            <rdfs:domain rdf:resource="&lea;Anwendungsservice"/>
204         <rdfs:range rdf:resource="&xsd;string"/>
        </owl:DatatypeProperty>
206

208     <!--
        ///////////////////////////////////////////////////////////////////////////////////
210     //
        // Classes
212     //
        ///////////////////////////////////////////////////////////////////////////////////
214      -->

216     <!-- http://www.softplant.de/lea#Anwendung -->

218     <owl:Class rdf:about="&lea;Anwendung">
            <rdfs:label xml:lang="de">Anwendung</rdfs:label>
220         <rdfs:label xml:lang="en">Application</rdfs:label>
            <rdfs:subClassOf rdf:resource="&lea;Software"/>
222         <istBestandteilDerArchitekturebene xml:lang="de">Anwendungsarchitektur</
                istBestandteilDerArchitekturebene>
            <description xml:lang="de">Eine Anwendung ist ein Softwareprodukt mit dem, wenn es installiert
                 ist, der Endanwender aus einem Unternehmen bzw. dessen Kunden in direkten Kontakt steht
                .</description>
224     </owl:Class>

226
        <!-- http://www.softplant.de/lea#Anwendungsservice -->
228
        <owl:Class rdf:about="&lea;Anwendungsservice">
230         <rdfs:label xml:lang="de">Anwendungsservice</rdfs:label>
            <rdfs:label xml:lang="en">Application Service</rdfs:label>
232         <rdfs:subClassOf rdf:resource="&lea;Service"/>
            <istBestandteilDerArchitekturebene xml:lang="de">Anwendungsarchitektur</
                istBestandteilDerArchitekturebene>
234         <description xml:lang="de">Der Anwendungsservice ist eine formale Schnittstelle ueber die eine
                 Geschaeftsanwendung angesprochen werden kann. Es handelt sich um einen Servicetyp ohne
                Umsetzung.
    </description>
236     </owl:Class>

238
        <!-- http://www.softplant.de/lea#Bebauungselement -->
240
        <owl:Class rdf:about="&lea;Bebauungselement">
242         <rdfs:label xml:lang="de">Bebauungselement</rdfs:label>
            <rdfs:label xml:lang="en">Business Support Element</rdfs:label>
```

```
244     </owl:Class>

246
        <!-- http://www.softplant.de/lea#BebauungselementImIst -->
248
        <owl:Class rdf:about="&lea;BebauungselementImIst">
250         <rdfs:label xml:lang="de">Bebauungselement im Ist</rdfs:label>
            <rdfs:label xml:lang="en">Current Business Support Element</rdfs:label>
252         <rdfs:subClassOf rdf:resource="&lea;Bebauungselement"/>
            <istBestandteilDerArchitekturebene xml:lang="de">Bebauungsplanung (vertikale Architektur)</
                istBestandteilDerArchitekturebene>
254         <description xml:lang="de">Ein Bebauungselement im Ist verbindet einen Anwendungsservice mit
                einer eindeutigen Kombination aus einer Rolle und einem Prozessschritt oder einer
                Organisationseinheit und einem Prozess.</description>
        </owl:Class>
256

258     <!-- http://www.softplant.de/lea#BebauungselementImSoll -->

260     <owl:Class rdf:about="&lea;BebauungselementImSoll">
            <rdfs:label xml:lang="de">Bebauungselement im Soll</rdfs:label>
262         <rdfs:label xml:lang="en">Target Business Support Element</rdfs:label>
            <rdfs:subClassOf rdf:resource="&lea;Bebauungselement"/>
264         <istBestandteilDerArchitekturebene xml:lang="de">Bebauungsplanung (vertikale Architektur)</
                istBestandteilDerArchitekturebene>
            <description xml:lang="de">Ein Geschaeftsunterstuetzungselement im Soll verbindet einen
                Anwendungsservice und Informationsservice mit einer eindeutigen Kombination aus einer
                Rolle und einem Prozessschritt oder einer Organisationseinheit und einem Prozess.</
                description>
266     </owl:Class>

268
        <!-- http://www.softplant.de/lea#Informationsobjekt -->
270
        <owl:Class rdf:about="&lea;Informationsobjekt">
272         <rdfs:label xml:lang="de">Informationsobjekt</rdfs:label>
            <rdfs:label xml:lang="en">Information Object</rdfs:label>
274         <description xml:lang="de">Ein Informationsobjekt ist ein virtuelles Artefakt zur Entkopplung
                von Objekten aus der Geschaeftswelt (Geschaeftsobjekte) und deren Abbildung mittels
                Informationstechnologie (Datenobjekte).</description>
            <istBestandteilDerArchitekturebene xml:lang="de">Informationsarchitektur</
                istBestandteilDerArchitekturebene>
276     </owl:Class>

278
        <!-- http://www.softplant.de/lea#Informationsservice -->
280
        <owl:Class rdf:about="&lea;Informationsservice">
282         <rdfs:label xml:lang="de">Informationsservice</rdfs:label>
            <rdfs:label xml:lang="en">Information Service</rdfs:label>
284         <rdfs:subClassOf rdf:resource="&lea;Service"/>
            <description xml:lang="de">Ein Informationsservice ist ein virtuelles Artefakt zur Entkopplung
                 von Services aus der Geschaeftswelt (Geschaeftsservices) und deren Abbildung mittels
                Informationstechnologie (IT-Services).</description>
286         <istBestandteilDerArchitekturebene xml:lang="de">Informationsarchitektur</
                istBestandteilDerArchitekturebene>
        </owl:Class>
288

290     <!-- http://www.softplant.de/lea#Organisationseinheit -->

292     <owl:Class rdf:about="&lea;Organisationseinheit">
            <rdfs:label xml:lang="de">Organisationseinheit</rdfs:label>
294         <rdfs:label xml:lang="en">Organizational Unit</rdfs:label>
            <description xml:lang="de">Eine Organisationseinheit ist eine organisatorische Unterteilung
                eines Unternehmens und hat eine oder mehrere ihr zugeordnete Mitarbeiter.</description>
```

```
296         <istBestandteilDerArchitekturebene xml:lang="de">Geschaeftsarchitektur</
                istBestandteilDerArchitekturebene>
        </owl:Class>
298

300     <!-- http://www.softplant.de/lea#Prozess -->

302     <owl:Class rdf:about="&lea;Prozess">
            <rdfs:label xml:lang="de">Prozess</rdfs:label>
304         <rdfs:label xml:lang="en">Business Process</rdfs:label>
            <description xml:lang="de">Ein Prozess beschreibt den Ablauf von Taetigkeiten und hat ein
                Ergebnis. Prozesse werden in Prozessaktivitaeten verfeinert.</description>
306         <istBestandteilDerArchitekturebene xml:lang="de">Geschaeftsarchitektur</
                istBestandteilDerArchitekturebene>
        </owl:Class>
308

310     <!-- http://www.softplant.de/lea#Prozessaktivitaet -->

312     <owl:Class rdf:about="&lea;Prozessaktivitaet">
            <rdfs:label xml:lang="de">Prozessaktivitaet</rdfs:label>
314         <rdfs:label xml:lang="en">Process Activity</rdfs:label>
            <description xml:lang="de">Eine Prozessaktivitaet beschreibt eine Aufgabe die in einem Prozess
                 die von einer Rolle zu erledigen ist. Auerdem stellt er eine Verfeinerung eines
                Prozesses dar.</description>
316         <istBestandteilDerArchitekturebene xml:lang="de">Geschaeftsarchitektur</
                istBestandteilDerArchitekturebene>
        </owl:Class>
318

320     <!-- http://www.softplant.de/lea#Rolle -->

322     <owl:Class rdf:about="&lea;Rolle">
            <rdfs:label xml:lang="de">Rolle</rdfs:label>
324         <rdfs:label xml:lang="en">Role</rdfs:label>
            <description xml:lang="de">Eine Rolle definiert bestimmte Aufgaben, Kompetenzen und
                Verantwortlichkeiten die mit dieser verknuepft sind. Ein Mitarbeiter nimmt eine oder
                mehrere Rollen im Unternehmen ein und somit auch die damit verknuepften Aufgaben,
                Kompetenzen und Verantwortungen.
326 </description>
            <istBestandteilDerArchitekturebene xml:lang="de">Geschaeftsarchitektur</
                istBestandteilDerArchitekturebene>
328     </owl:Class>

330
        <!-- http://www.softplant.de/lea#Service -->
332
        <owl:Class rdf:about="&lea;Service">
334         <rdfs:label xml:lang="de">Service</rdfs:label>
            <rdfs:label xml:lang="en">Service</rdfs:label>
336     </owl:Class>

338
        <!-- http://www.softplant.de/lea#Software -->
340
        <owl:Class rdf:about="&lea;Software">
342         <rdfs:label xml:lang="de">Software</rdfs:label>
            <rdfs:label xml:lang="en">Software</rdfs:label>
344     </owl:Class>
    </rdf:RDF>
346
    <!-- Generated by the OWL API (version 3.5.1) http://owlapi.sourceforge.net -->
```